AD-A060 850 HONEYWELL INC MINNEAPOLIS MINN SYSTEMS AND RESEARCH --ETC F/G 17/5
PROTOTYPE AUTOMATIC TARGET SCREENER.(U)
SEP 78 D E SOLAND, P M NARENDRA, R C FITCH DAAK70-77-C-0248
UNCLASSIFIED 78SRC54-3 NL

1 OF 2
AD
A060850

PROTOTYPE AUTOMATIC TARGET SCREENER

By

D.E. Soland
P.M. Narendra
R.C. Fitch
D.V. Serreyn
T.G. Kopet

Honeywell

SYSTEMS & RESEARCH CENTER
2600 RIDGWAY PARKWAY
MINNEAPOLIS, MINNESOTA 55413

September 1, 1978

Quarterly Progress Report 1 April–30 June 1978

Prepared for

U.S. Army Mobility Equipment
Research and Development Command,
Night Vision and Electro-Optics Laboratory
Fort Belvoir, Virginia 22060

"The views, opinions, and/or findings contained in this report are those of the authors and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentations."

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (WHEN DATA ENTERED)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOV'T ACCESSION NUMBER	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (AND SUBTITLE) PROTOTYPE AUTOMATIC TARGET SCREENER		5. TYPE OF REPORT/PERIOD COVERED Quarterly Progress Report, no. 3 1 Apr [illegible] to 30 June [illegible] 78 6. PERFORMING ORG. REPORT NUMBER 78SRC54-3
7. AUTHOR(S) D. E. Soland, D. V. Serreyn P. M. Narendra, T. G. Kopet R. C. Fitch		8. CONTRACT OR GRANT NUMBER(S) DAAK70-77-C-0248
9. PERFORMING ORGANIZATIONS NAME/ADDRESS Honeywell Systems and Research Center 2600 Ridgway Parkway Minneapolis, Minnesota 55413		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 1E263710DK70 14 010CJ
11. CONTROLLING OFFICE NAME/ADDRESS Night Vision and Electro-Optics Laboratory Fort Belvoir, Virginia 22060		12. REPORT DATE Sep[illegible], 78 13. NUMBER OF PAGES 108
14. MONITORING AGENCY NAME/ADDRESS (IF DIFFERENT FROM CONT. OFF.) 109 p.		15. SECURITY CLASSIFICATION (OF THIS REPORT) Unclassified 15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (OF THIS REPORT)

Approved for public release, distribution unlimited.

17. DISTRIBUTION STATEMENT (OF THE ABSTRACT ENTERED IN BLOCK 20, IF DIFFERENT FROM REPORT)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (CONTINUE ON REVERSE SIDE IF NECESSARY AND IDENTIFY BY BLOCK NUMBER)

Infrared	Target recognition	Image enhancement
FLIR	Pattern recognition	
Target cueing	Image processing	
Target screening	Real-time	

20. ABSTRACT (CONTINUE ON REVERSE SIDE IF NECESSARY AND IDENTIFY BY BLOCK NUMBER)

This report is the third quarterly progress report for contract DAAK70-77-C-0248, Prototype Automatic Target Screener. The objective of the effort is to design an automatic target screener to be used with thermal imaging systems employing common module components.

HD-168 REV 11/74

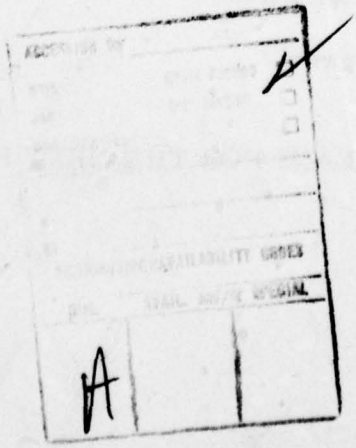
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 55 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (WHEN DATA ENTERED)

402349

CONTENTS



CONTENTS (concluded)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (continued)

LIST OF ILLUSTRATIONS (concluded)

LIST OF TABLES

SECTION I

INTRODUCTION AND SUMMARY

This is the third quarterly technical progress report for contract number DAAK70-77-C-0248, Prototype Automatic Target Screener (PATS). The first two quarterly reports documented the Phase I design study. This report covers the first part of Phase II. Phase II is a seventeen-month effort which includes detailed system design, fabrication, integration, and evaluation and testing. The period covered by this report is 1 April to 30 June 1978.

The program objective is to produce a design for an automatic target screener. The screener will reduce the task loading on the thermal imager operator by detecting and recognizing a limited set of high priority targets at ranges comparable to or greater than those for an unassisted observer. A second objective is to provide enhancement of the video presentation to the operator. The image enhancement includes (1) automatic gain/brightness control to relieve the operator of the necessity to continually adjust the display gain and brightness controls, and (2) DC restoration to eliminate artifacts resulting from AC coupling of the infrared (IR) detectors.

The image enhancement portion of PATS will consist of circuitry to operate on the Common Module FLIR (MODFLIR) video output signal. The circuitry will provide global gain and bias control in the form of

feedback to the MODFLIR to maintain the signal within the dynamic range of the electro-optical multiplexer.

Image enhancement will also include local area gain and brightness control to enhance local variations of contrast and compress the overall scene dynamic range to match that of the display. This circuitry has been completed and examples of its performance on videotaped thermal image data were included, along with the circuit description, in the first quarterly report.

The third image enhancement circuit is for DC restoration to eliminate the streaking associated with loss of line-to-line correlation on the displayed image because of the AC coupling of the detector channels.

This report consists of five sections. Section II describes the final results of the classifier design and optimization study. Specifically, we have improved the clutter rejection classifier by adding a new feature and reducing the total number of features. Also, we have replaced the cascade threshold linear classifier by a k-nearest neighbor classifier for new scenarios, different sensors, etc. while somewhat increasing the computations required for recognition. Final recognition accuracy obtained on the design data set was 78.4 percent.

Section III presents the results of the software design effort. This includes detailed specifications for each of the software functions required. Sections IV and V include descriptions of the detailed hardware design effort to date at the function and systems level, respectively. Section VI describes the planned effort for the next three-month reporting period.

SECTION II

CLASSIFIER OPTIMIZATION

In the last quarterly progress report we presented the results of a preliminary two-stage classifier for clutter screening and target classification. Both stages used cascaded threshold classifiers, which were comprised of a set of linear discriminants designed with the training samples from the target classes and representative clutter. During this period, we optimized the clutter rejection and the target recognition classifiers in the following ways:

1. We improved the clutter rejection classifier by adding a new feature based on edge and eliminating all but the most essential remaining features. We made this cascaded threshold classifier structurally simple in order to make it more robust when used with independent data sets.

2. We implemented a k-nearest neighbor (kNN) classifier for the recognition stage. This replaces the cascaded threshold linear classifier described in the previous report.

The kNN classifier for recognition makes it easy to "train" the classifier with new data because the training process consists of merely storing the prototypes. This feature is important when new target classes are added, for training in situ, and for varying scenarios. The main drawbacks of

this type of classifier have been that it requires expensive distance calculations and storage of a large number of training samples. Both drawbacks now appear surmountable with the CPU 1's fast multiplier/adder.

In this section we present the results of the two new classifiers and compare their performance with the preliminary classifiers discussed in the previous report. Implementation of these classifiers in CPU 1 software in PATS is also discussed.

CLUTTER REJECTION CLASSIFIER

The preliminary clutter rejection classifier, previously described, used 14 features in all and had an involved hierarchical structure of cascaded thresholds five stages deep. We felt that this classifier could be simplified by pruning the nonessential and correlated features and reducing the number of classification steps.

Considering the features one at a time, we examined the scatter of targets and nontargets. We discarded features which showed no clustering of targets and nontargets. Several of the features were highly correlated, so we discarded features that were correlated with the chosen set. This reduced the feature set to three features:

1. Average target intensity
2. Average target contrast
3. Total area

In addition, a clutter prescreening stage, which consists of straight thresholds on the target area and the bright count, was implemented to throw out any pathological clutter. This rejects all targets less than 43 pixels* in area and possessing a bright count of less than 750. The bright count is the number of thresholded (hot/cold) pixels in the object normalized with respect to the area of the extracted objects. This fraction is multiplied by a factor of 1000. A bright count of 750, therefore, implies that 75 percent of the extracted object was either below or above the cold or hot thresholds, respectively. Therefore, small clusters of isolated thresholded points are rejected by this prescreening stage.

To strengthen the clutter rejection classifier further, we defined two new features based on edges (not the edge count feature described previously, which is of little value). These are based on the fact that targets usually possess edges on both sides whereas most clutter objects extracted have a significant edge on one side but none on the other. These features are called the left and right edge features, respectively, and are defined as follows:

Right (left) edge feature = Number of intervals in the object for which an edge exists at the right (left) of the interval/number of intervals.

A set of 40 frames was reprocessed and the above new features were computed on each extracted object and appended to the previously extracted features. There were 865 objects present (186 targets and 679 clutter objects).

*One pixel corresponds approximately to one IFOV for this sensor.

Figures 1a and 1b show the scatter plot of the two new features for targets and clutter objects, respectively. From this we see that thresholds of approximately 600 on each axis succeed in removing a substantial amount of clutter with minimal loss of targets. However, a large number of clutter objects are in the vicinity of the target samples. Nevertheless, these two features have proved to be good independent clutter rejectors. Therefore, our refined clutter rejection scheme includes these two features in addition to the three features mentioned before.

A linear discriminant classifier was attempted using all six features and one discriminant to separate the two classes (targets, clutter). The results were unsatisfactory, implying that the target and clutter classes were not unimodal Gaussian, which is a requirement of the linear discriminant classifiers. Therefore, a hierarchical classification scheme was developed using these five features. This classifier retains the simplicity but has a better clutter rejection performance.

Figure 2 shows the structure of the optimized clutter classifier. It consists of a sequence of cascaded thresholds on the six features: bright count, left and right edge features, average target intensity, average target contrast, and target area. No linear discriminant computations (involving multiplies) are used. This is desirable in a real-time clutter screening classifier designed to handle a large number of objects. Referring to Figures 1a and 1b we see that:

- The easy clutter classification stage removes 378 out of 679 clutter objects. This stage uses the bright count and the total area.

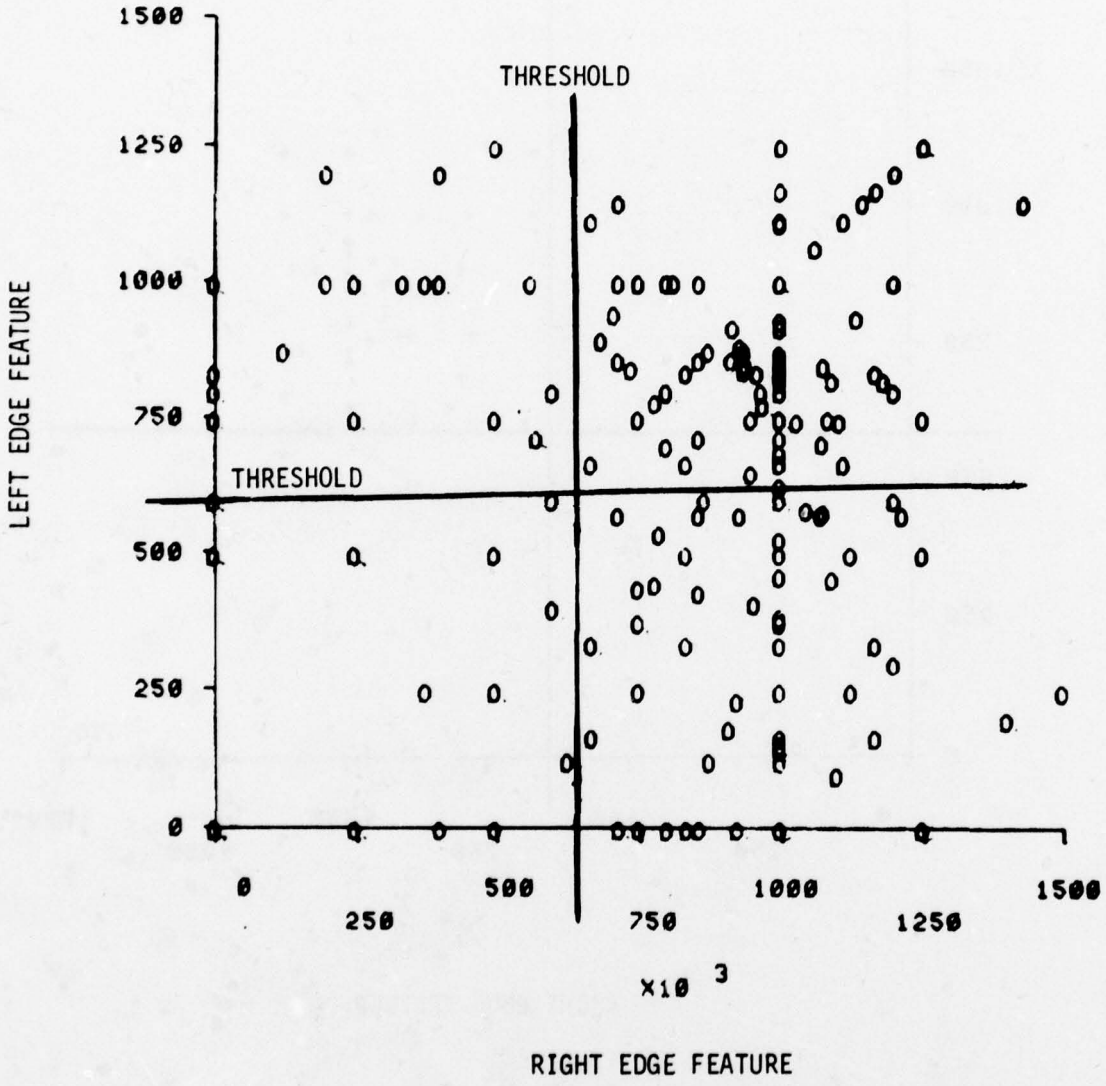


Figure 1a. Plot of Edge Features for Clutter Objects

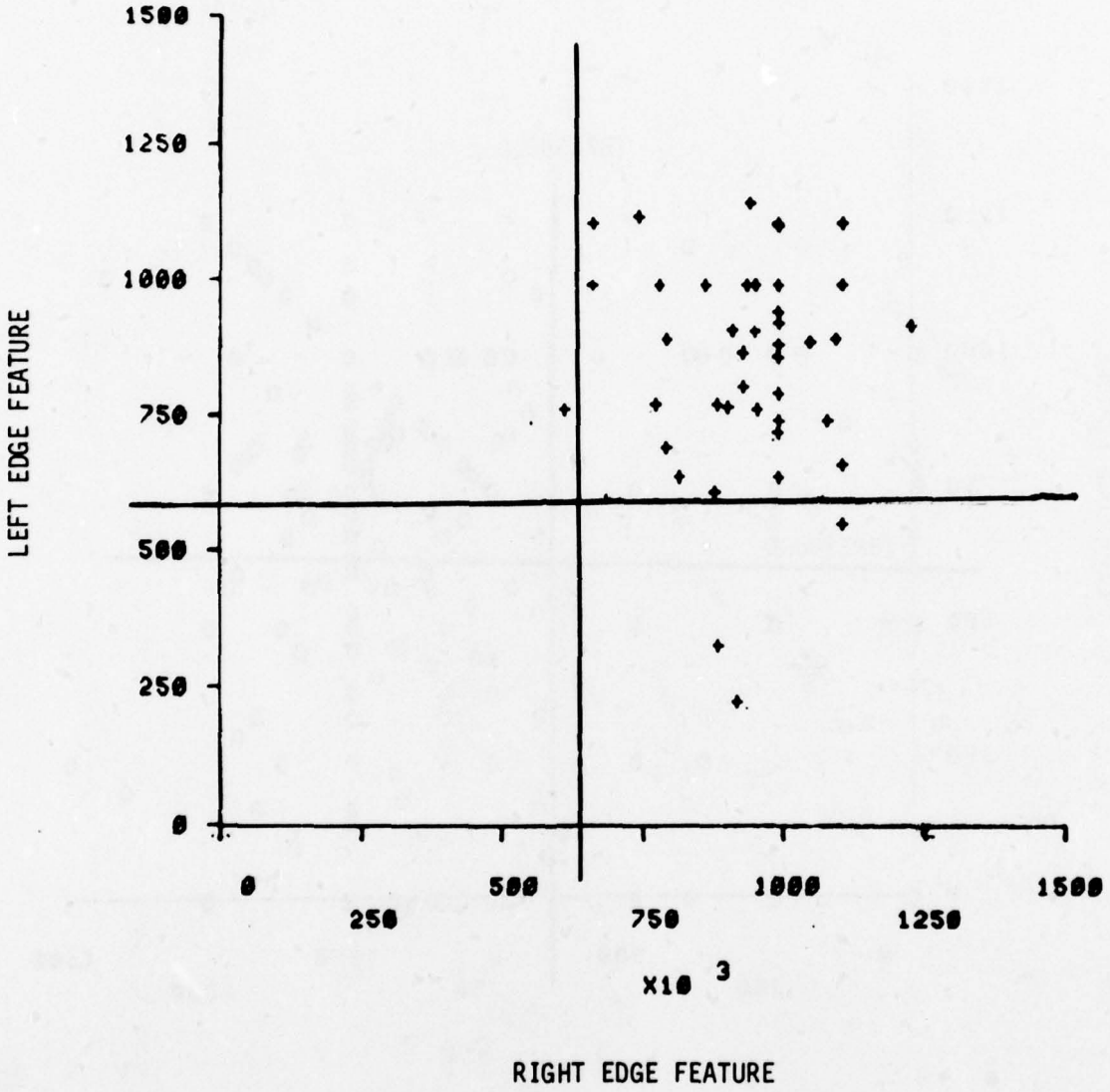


Figure 1b. Plot of Edge Features for Targets

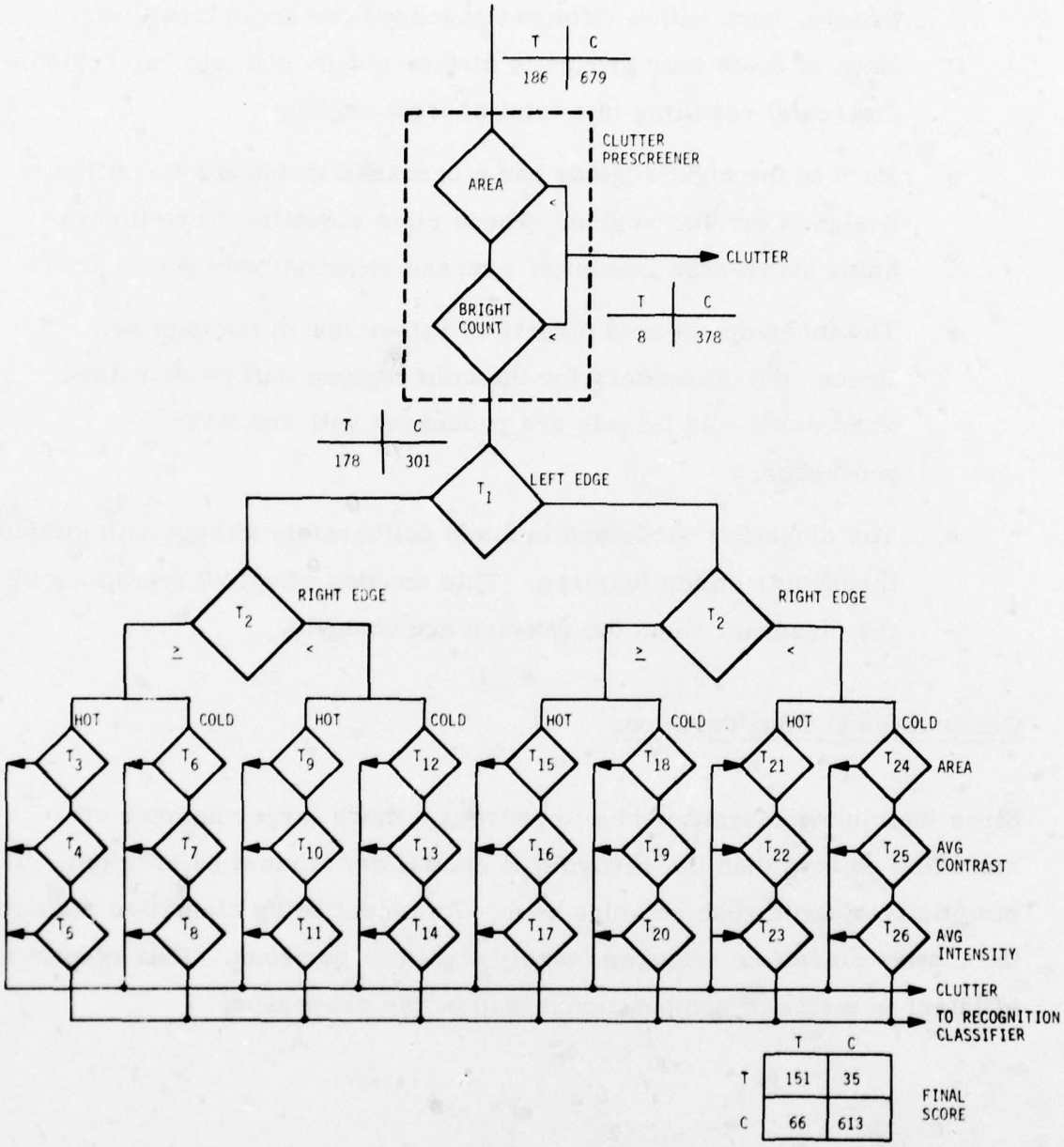


Figure 2. Clutter Classifier Structure

- The right and left edge features partition the data into four groups, each with a different cascaded threshold classifier. Each of these four groups is further subdivided into two regions (hot/cold) resulting in a total of eight regions.
- Each of the eight regions has a cascaded threshold classifier designed for that region. These eight classifiers use thresholds on average intensity, average contrast, and object area.
- The incidence of cold targets is rather low in the data set. Hence, the classifiers for the cold regions will be retrained when more cold targets are processed with the PATS processor.
- The classifier structure is made deliberately simple with simple thresholds on the features. This enables effective retraining of the classifier when the sensors are changed.

Computational Considerations

Since the clutter classifier has to process a much larger number of extracted objects than the recognition classifier, it must be efficient. The simplicity of the threshold helps here. An object being classified through the clutter classifier requires merely eight comparisons. This is extremely efficient in terms of computational load on the processor.

RECOGNITION CLASSIFIER

In the previous report[1] we presented a preliminary recognition classifier for two target classes (tanks and APCs). This classifier used cascaded linear discriminants. Because the k-nearest neighbor classifier appears to be more attractive for the PATS recognition application, we tested the approach with training data acquired from the previously processed images. In this subsection we describe the kNN classifier approach employed, the feature selection procedure to select the "best" subset of features, and the results of classification with the training data. Implementation considerations include a description of a fast algorithm to compute the k-nearest neighbors.

The kNN Classifier

Training samples (feature vectors for objects whose real category is known) and their categories are stored. A new sample vector is classified as follows. Its distances to all the stored training samples are found, and its k-nearest neighbors among the training samples are determined. The new sample is assigned to the class to which the majority of its k neighbors belong. When the number of classes is two, k is usually chosen odd to avoid ties.

---

[1] Duane Soland, et al., "PATS Quarterly Progress Report," Contract No. DAAK70-77-C-0248, Honeywell Systems and Research Center, Minneapolis, Minnesota, June 15, 1978.

The advantages of the kNN classifier are as follows:

- The classifier is easy to train: we just store the training samples. When new target classes/scenarios are to be accommodated, we simply add new training samples representing the new scenario to the training set.
- The classifier is nonparametric: the distributions for each class need not be linearly separable. In fact, they can possess any arbitrary shape and can be multimodal. This is attractive because we can make the ground truth (true classification) aspect independent in the training process. In fact, we only have to specify the target class (tank, APC, etc.). Of course we still have to ensure that all aspects are adequately represented in the training data.

The disadvantages of the kNN approach are as follows:

- Because the process is nonparametric, a large number of training samples is necessary. This number grows very quickly with the number of features being used.
- Storage of training samples requires N x (n + 1) words of memory to store N vectors of dimension n, along with the ground truth.
- Computation of the nearest neighbors is expensive. The brute-force approach requires the calculation of all N distances to classify the new sample. This requires N x n multiply-adds/classification.

The drawbacks can be at least partially overcome by:

1. Feature selection to limit the number of features to the minimum subset that yields acceptable recognition.
2. Using clever techniques to find the nearest neighbors of a new sample without actually calculating all N distances.
3. The use of high-speed memory and the 115 nsec multiplier/add in the PATS processor.

We developed kNN classifier software on the SDS9300 computer to evaluate the kNN approach. Because of the emphasis on analysis, the software was designed to be interactive and flexible. For example, the program takes as input the specific features to be used, the parameter k, and the training vector tape that contains all the features on all the objects extracted with the ground truth. The program outputs confusion matrices--a summary of how samples from each class were assigned by the k-nearest neighbor vote. When used to measure performances on the training data, the classifier program does not count the sample being tested as one of its own neighbors. If it were otherwise, with k = 1, we would get 100 percent correct classification performance on the training data! In each run made, the program gives the classification performance with all values of k from 1 to 10. In this way, we monitor the effect of k on the performance and select the best k for a given feature set.

Feature Subset Selection

The objective of this effort was to find the minimum number of features from the three moment feature sets, each with six usable moments (μ_{11}, μ_{20}, μ_{12}, μ_{21}, μ_{03}, μ_{30}). (μ_{00} is used for normalization, and μ_{02} and μ_{20} are perfectly negatively correlated after normalization. Hence μ_{00} and μ_{02} are not used.) In these experiments 227 objects (143 tanks and 84 APCs) were used for data.

Which Moment Type?--Our first feature selection step was to choose one family of moments from the intensity-, silhouette-, and boundary-derived moments. Accordingly, we tested the kNN classifier with the six moments from each class. In each case, all values of k = 1, ..., 10 were tried, and the results with the best value are presented. Table 1 summarizes these results. The percentage error in Table 1 was computed as the percent fraction of the total misclassified samples to the total number of samples. We note that the three moment types are not markedly different in performance from one another. The boundary moments yielded the lowest error rate (21.6 percent), but we chose the intensity moments (at 22.5 percent error rate). Intensity moments tend to be less susceptible to noise in segmentation than boundary moments because the boundaries are less heavily weighted.

Optimum Moment Subset Selection

There are six usable intensity moments: μ_{11}, μ_{12}, μ_{21}, μ_{20}, μ_{30}, μ_{03}. Our next step was to find the minimal subset which gave almost the same performance as all six moments. We sought to reduce the number of

TABLE 1. COMPARISON OF MOMENT FEATURE TYPES
(Two classes, kNN classifier)

Moment Type ($\mu_{11}, \mu_{12}, \mu_{20}, \mu_{21}, \mu_{03}, \mu_{30}$)	Optimum k	Percent Error
Intensity	8	22.5
Boundary	7	21.6
Silhouette	7	24.2

features because (1) the smaller the dimensionality of the feature space, the fewer the training samples we will need for robust classification, (2) the computation requirements for the kNN classifier increase linearly with the number of features when the direct nearest neighbor computation is used, and (3) when using the "clever" techniques for nearest neighbor computation, this increase in computation with the number of features is even more severe, as we will see later.

There are $N!/M!(N - M)!$ unique ways in which we can choose M features from a collection of N features. This combinatorial is denoted by the conventional notation $\binom{N}{M}$. Thus, $\binom{6}{2}$ denotes the number of two feature subsets that are in the six features:

$$\binom{6}{2} = \frac{6!}{2!(6-2)!} = \frac{6 \times 5 \times 4 \times 3 \times 2 \times 1}{(2 \times 1) \times (4 \times 3 \times 2 \times 1)} = 15$$

Similarly,

$$\binom{6}{3} = 20, \quad \binom{6}{4} = 15, \quad \text{and} \quad \binom{6}{5} = 5$$

By evaluating the kNN error rates with all subsets of a given size (2, 3, 4, 5), we found the best subset of size 2, 3, 4, and 5 and the corresponding k (the number of nearest neighbors used). Table 2 summarizes these results. Fewer features generally tend to yield higher error rates although the relationship is by no means strictly monotonic as we can see from Table 2. From this analysis, we have chosen the best subset of four features (μ_{11}, μ_{02}, μ_{21}, μ_{03}) as the features to be used in the recognition classifier. In fact, these features yield an error rate of 21.6 percent, which is lower than the error (22.5 percent) with all six features. The confusion matrix with these four features is given in Table 3.

This results in a total correct classification of 178/227 = .784, or 78.4 percent, which is somewhat better than the corresponding performance of 75.3 percent obtained with the CTC classifier reported in the Second Quarterly Progress Report.

Choice of Distance Measure

In the above kNN classifier, we used the usual Euclidean distance to define the neighborhood. The Euclidean distance between two n-dimensional feature vectors X and Y is given by

TABLE 2. RESULTS OF FEATURE SUBSET SELECTION

	Size of Feature Subset				
	2	3	4	5	6
Best Feature Subset	μ_{11}, μ_{02}	$\mu_{11}, \mu_{02}, \mu_{12}$	$\mu_{11}, \mu_{02}, \mu_{21}, \mu_{03}$	$\mu_{11}, \mu_{02}, \mu_{21}, \mu_{12}, \mu_{03}$	$\mu_{11}, \mu_{02}, \mu_{21}, \mu_{12}, \mu_{03}, \mu_{30}$
Error with Best Subset	32.6%	36.1%	21.6%	25.1%	22.5%
Optimum k for Best Subset	2	2	7	4	7
Number of Feature Subsets Possible	15	20	15	6	1

TABLE 3. PERFORMANCE OF THE BEST FOUR MOMENT SET (kNN classifier)

		Assigned Class	
		Tanks	APCs
True Class	Tanks	112	31
	APCs	18	66

$$d_E^2(X, Y) = \sum_{i=1}^{n} (x_i - y_i)^2$$

where

$$X = (x_1 \ldots x_n)^T$$

$$Y = (y_i \ldots y_n)^T$$

This distance measure involves computation of squares, which is a time-consuming arithmetic multiply step. Other measures such as the "city block" or "taxi-cab" distance are easier to compute.

The "city block" distance between sample vectors X and Y is given by

$$d_c(X, Y) = \sum_{i=1}^{n} |x_i - y_i|$$

where the sum of absolute differences substitutes for the sum of squared differences in the Euclidean metric. This metric avoids the multiplies and is therefore easier to compute.

We computed the kNN error using the city block distances for the first six subsets of four features from the six intensity moment features. The results are shown in Table 4, which shows the confusion matrixes for each case as well. From these runs, we see that the city block distance performs almost as well as, but is almost consistently behind, the Euclidean distance. Timing analysis in the software section shows that, in the PATS processor, there is very little to be gained by using the city block distance. This is in part a consequence of the 115 nsec hardware multiplier/adder used in the PATS processor.

Fast NN Computation

The computer program for nearest neighbor analysis in our simulation uses the brute force approach to find the nearest neighbors. That is, to find the nearest neighbors of a given test sample among a training set of N samples, we compute all N distances. Recently, clever schemes for finding the nearest neighbors without evaluating all N distances have been proposed.[2, 3] These algorithms are based on ordering the training sample set in such a

---

[2] K. Fukunaga and P. M. Narendra, "A Branch and Bound Algorithm for Computing k-Nearest Neighbors," IEEE Transactions on Computers, July 1975.

[3] J. H. Friedman, et al., "An Algorithm for Finding Best Matches in Logarithmic Expected Time," ACM TOMS, Vol. 3, September 1977.

TABLE 4. COMPARISON OF THE EUCLIDEAN AND CITY BLOCK METRICS FOR THE kNN CLASSIFIER

Feature Set	Metric Used			
	Euclidean		City Block	
	Confusion Matrix	Percent Error	Confusion Matrix	Percent Error
$\mu_{11}, \mu_{02}, \mu_{30}, \mu_{21}$	113 30 21 63	22.5	111 32 22 62	23.8
$\mu_{11}, \mu_{02}, \mu_{30}, \mu_{12}$	119 24 32 52	24.7	115 28 36 48	28.2
$\mu_{11}, \mu_{02}, \mu_{30}, \mu_{03}$	120 23 38 46	26.7	120 23 34 50	25.1
$\mu_{11}, \mu_{02}, \mu_{21}, \mu_{12}$	119 24 31 53	24.2	108 35 21 63	24.7
$\mu_{11}, \mu_{02}, \mu_{21}, \mu_{03}$	112 31 18 66	21.6	110 33 19 65	22.9
$\mu_{11}, \mu_{02}, \mu_{12}, \mu_{03}$	114 29 30 54	26.0	114 29 31 53	26.4

way that we do not have to consider all samples when finding the nearest neighbors. Thus, there is some overhead in the initial preprocessing of the training samples. But this needs to be done only once for a given set of training data. The utility of these algorithms for PATS depends on several considerations. A characteristic of these algorithms is that their relative efficiency increases with the number of training samples and decreases rapidly with the dimensionality, the number of features used.

This was one of the reasons why we reduced the feature subset to the smallest size possible without sacrificing recognition performance.

Both Friedman's algorithm and the Fukunaga-Narendra algorithm are very similar in performance and principle. But Friedman's paper contains exhaustive simulation results which let us predict with reasonable accuracy the computational effort involved in implementing the approach in the PATS processor. Therefore, we are implementing Friedman's scheme in the PATS processor.

A preliminary analysis of nearest neighbor to computation requirements was made for the PATS processor. A four-feature Euclidean distance computation requires 3 μsec. Assuming 1000 training samples, the classification of one object requires 3 msec when the brute force technique is used, ignoring overhead. For 10 objects, this is 30 msec. But all other functions in the processor-object extraction, moment computation, etc., take up a total of 28 msec (see Section V). In this light, the 30 msec requirement for the nearest neighbor computations seems to be excessive although the total processing time (68 msec) would still be under the 100 msec limit allowed when overhead is ignored. With 100 percent overhead, we would be overloading the processor.

Using the fast NN algorithm, we can reduce the number of distance computations down to 100 even when the training set contains 1000 samples. This implies a saving of a factor of 10 and is well worth pursuing. Therefore, 10 objects can be classified in 3 msec. Even with 100 percent overhead, this is 6 msec.

Another feature of these algorithms is that the computation is almost independent of the number of training samples. A training set of 2000 samples, for instance, would require the same number of distance computations (100) but perhaps a little more overhead. Therefore, if occasion demands, we can accommodate a number of mission scenarios in the same training set by expanding its size.

The Fast NN Algorithm--We will give a brief description of Friedman's algorithm here. The training samples are first preprocessed, which consists of dividing them into successively smaller subgroups. The hierarchic subdivision is represented as a binary tree structure. Figure 3 shows such a tree for 256 samples. Each group is successively divided into two subgroups of equal size, until the terminal stage. There are 16 groups with 16 samples each at the terminal stage. Each group is divided by choosing the axis (feature) with the largest variance or spread and splitting the axis at the median of the distribution of the sample on that axis. This tree and associated bounds are stored in a simple data structure. This completes the preprocessing step.

When a new sample vector needs to be classified, we have to find its k-nearest neighbors from the training set. Starting from the top of the tree, each group is examined to see if the group might contain the nearest neighbors of the new sample. The efficiency of the algorithm lies in detecting groups that cannot contain the neighbor of a point. We can then eliminate them from the search. This test to assure that a group of samples at any level of the tree cannot contain the k-nearest neighbors is called the "ball overlaps bound" test. Figure 4 illustrates this test with a two-dimensional

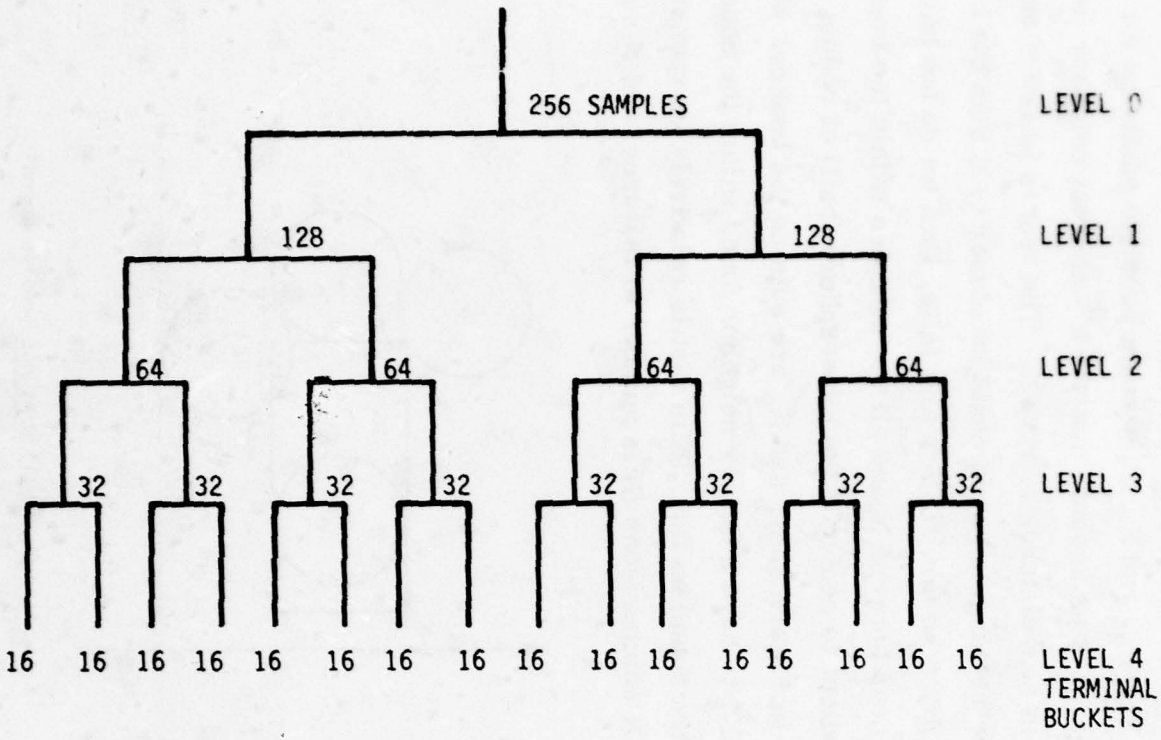
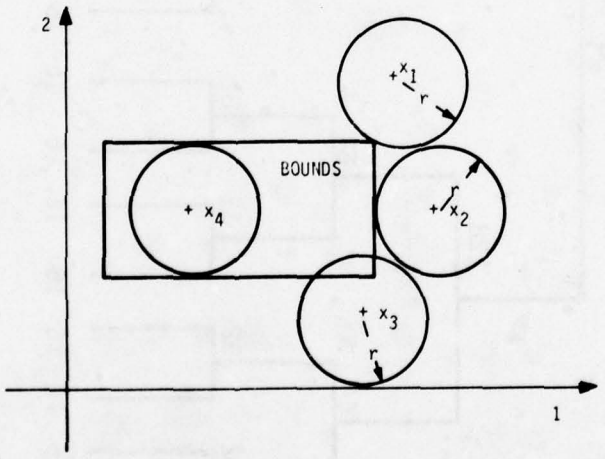


Figure 3. Hierarchical Tree Defining the Grouping of 256 Training Samples for the Fast NN Algorithm

example (two features: 1 and 2). The rectangle shows the bounds along dimensions 1 and 2 of all the samples contained in a given group. X is the new sample (X_1, X_2, X_3, and X_4 show some possible positions for X relative to the bounds). Let us assume that the k^{th} nearest neighbor found so far in the search is at a distance r from X. The test is to see if any samples within the rectangular area could be closer to X than the k^{th} nearest neighbor found so far. If this test fails, then we do not have to compute the distances between X and all the samples within the bounds. This test is equivalent to seeing if the hypersphere (ball) of radius r overlaps the bounds. In Figure 3, X_1 and X_2 are outside the bounds, whereas it is possible for X_3 to have a nearer neighbor found within the bounds than the nearest neighbor found so far. This test is relatively inexpensive and requires at most N comparisons (N is number of features) and N squaring operations.



x_1 and x_2 - ball does not overlap bounds

x_3 - ball overlaps bounds

x_4 - ball completely within bounds

Figure 4. Ball overlaps Bounds and Ball within Bounds Tests

If the "ball" is completely within bounds as shown by X_4, then the nearest neighbor of X_4 has to be in this group and this group only. Therefore, if this test is satisfied for any group, we terminate the search, and the current k-nearest neighbor list contains the nearest neighbors we want. If the ball overlaps the bounds (as with X_3 in Figure 3), then we descend down the tree and test one of the nodes under the current node. When a terminal node is encountered, we find the distances from X to all the members (16) of that terminal node and update the list of nearest neighbors and their distances.

The preprocessing--building the tree and the bounds for each node--will be done off line in the "training process." The tree and the training samples will be stored in the PATS memory. The storage required (in excess of the storage required for the training samples themselves) is not very significant. For 1024 samples, 16 samples/terminal bucket, we have 64 terminal nodes. This results in a binary tree with 64 + 32+ 16 + 8 + 4 + 2 + 1 = 127 nodes. Each node stores 2 x N bounds (defining the rectangular area in Figure 3) where N is the number of features. With four features, the storage associated with the tree becomes 4 x 2 x 127 = 1016 words. It is also necessary to identify the sample vectors that belong to each terminal bucket. This requires 64 x 16 = 1024 words of memory. Hence the total storage associated with the algorithm will be 1016 + 1024 = 2040 words. Storing the samples themselves requires another 4 x 1024 = 4096 words of storage and 1024 words for the ground truth.

A detailed analysis of the algorithm steps will be made in the next reporting period prior to coding the microinstructions. At this time we feel that the fast NN algorithm is simple enough to be implemented in the PATS processor.

INTERFRAME ANALYSIS

The outputs of the clutter rejection and recognition classifiers are a set of symbols and coordinates representing decisions on all extracted objects in a frame. Preceding and succeeding frames will have similar symbolic representations of the decisions for those frames. After registration and correlation of each symbol to a symbol on the previous frame, the result will be a variable length string of symbols representing a sequence of decisions on a given extracted object. For example, the symbol string

TTAC TT

for a sequence of seven frames may represent a decision sequence where T = "tank," A = "APC," C = "clutter," and a blank represents a miss (object not detected).

The decision displayed to the operator will be more reliable if it is based on the multi-frame decision sequence than if it is for a single frame. The question, then, is how to process the information represented by the symbol string to provide the best decision.

One approach would be to store the feature vectors for each of the extracted objects over a sequence of frames and compute a "smoothed" feature vector for each frame. The smoothed features would then be used to compute a more reliable decision for each frame. However, this approach would require excessive memory and computation, would not necessarily provide a more reliable decision, and would not resolve the problem of ambiguities resulting from different decisions on successive frames for the same object.

An alternate approach, and the recommended one, is based on Bayes' theorem.

Decision Smoothing by Bayes' Theorem

Let A_1 represent the true event "tank," A_2 represent "APC," etc. The corresponding decisions will be the events T, A, etc. By Bayes' theorem, the posteriori probability of the presence of a tank, given the decision "T," is given by

$$P(A_1/T) = \frac{P(T/A_1)P(A_1)}{P(T)}$$

where $P(A_1)$ is the a priori probability of the presence of a tank and $P(T/A_1)$ is the conditional propability that the target screener will correctly label the tank.

Similarly,

$$P(A_2/T) = \frac{P(A_2/T)P(A_2)}{P(T)}$$

is the probability that the object labeled "T" is actually an APC, where again $P(A_2)$ is the a priori probability that an APC is present and $P(T/A_2)$ is the probability that the target screener will incorrectly label the APC as a "T."

The conditional probabilities $P(T/A_1)$, $P(T/A_2)$, etc. are the entries in the confusion matrix resulting from the classifier training. The priori probabilities $P(A_1)$, $P(A_2)$, etc. are determined by the engagement scenario and can be entered by the commander in the field.

We can write the probability P(T) in terms of these known quantities:

$$P(T) = P(T/A_1)P(A_1) + P(T/A_2)P(A_2) + \ldots + P(T/A_N)P(A_N)$$

where N is the number of classes for which the classifier is designed.

The approach to decision smoothing is to apply Bayes' theorem recursively to each decision in a string. Thus, the a posteriori probability of a given target class in one frame becomes the priori probability for that class in the next. A simple example will illustrate the approach.

Suppose we have a two-class target screener with the possible events being A_1 = "tank" and A_2 = "not tank." Further suppose that the probability of correctly labeling a tank is $P(T/A_1) = 3/4$. Then the probability of a false alarm is $P(T/A_2) = 1/4$. Also, let the a priori probability of a tank presence be 1/4. Then we would compute the following sequence of a posteriori probabilities for a string of decisions TTTT:

FRAME NUMBER	POSTERIORI PROBABILITY
1	$P(\text{tank/T}) = \dfrac{3/4 \cdot 1/4}{3/4 \cdot 1/4 + 1/4 \cdot 3/4} = 1/2$
2	$P(\text{tank/TT}) = \dfrac{3/4 \cdot 1/2}{3/4 \cdot 1/2 + 1/4 \cdot 1/2} = 3/4$
3	$P(\text{tank/TTT}) = \dfrac{3/4 \cdot 3/4}{3/4 \cdot 3/4 + 1/4 \cdot 1/4} = 9/10$
4	$P(\text{tank/TTTT}) = \dfrac{3/4 \cdot 9/10}{3/4 \cdot 9/4 + 1/4 \cdot 1/10} = 27/28$

Note in this example that the string of correct decisions increases the probability monotonically of the correct decision being displayed. In general, if incorrect decisions are interspersed in a string of correct ones, the sequence may not be monotonic and it will take a larger number of frames to achieve a given confidence level. A low priori target probability will also require a longer sequence of correct decisions to achieve a given confidence level.

The advantages of this approach are as follows:

1. It is computationally simple and storage requirements are minimal.
2. It optimally uses the available information.
3. Target screener performance can be tuned to the operational scenario in the field by specifying the a priori probability of encountering each target class.
4. The decision can be displayed with a specified confidence level by thresholding the a posteriori probabilities.

Priority Target Mode

The Bayes' a posteriori probability approach provides a means of implementing the priority target mode of operation, where two of the five target classes are of much greater interest and the false alarm rate must be significantly lower. As noted above, two ways in which this can be accomplished are by specification of a priori target probabilities and by specification of decision display confidence thresholds. A third modification is simply to relabel the stored training samples in the nearest neighbor

classifier so that all non-priority targets are, in effect, treated as clutter. This relabeling will change the confusion matrix and consequently the conditional probabilities used in Bayes' theorem. Training data are not yet available for the priority target classes so this concept cannot be evaluated.

SECTION III

SOFTWARE DESIGN

The PATS implementation software has been summarized in a previous report.[1] What follows is a presentation of essentially the same information with some additions and organizational modifications. The software descriptions are given in a modular input/function/output format and tend to emphasize the data structures used by the PATS algorithms.

The major difference between this and the previous report is the addition of the k-nearest neighbor recognition classifier. This classifier is discussed in detail in Section II, and some details relevant to its implementation are covered here.

PATS software functions, as before, are partitioned between two processors, CPU 1 and CPU 2. CPU 1 is a high speed, purely microprogrammable processor of special design which bears most of the PATS real-time computational load, whereas CPU 2 is an off-the-shelf, medium speed, mini or microcomputer whose principal function is to interface the PATS system to external peripherals of standard design (i.e., mag tape, floppy disks, etc.). Functional descriptions of the various software modules making up both processors are given below, with emphasis on module interplay and data structure manipulation.

CPU 1 SOFTWARE

The software for CPU 1 can be broken down into two categories: bin generation and bin classification. Bin generation segments the FLIR image into two-dimensional objects by matching intervals from adjacent scan lines. Bin classification processes each of these objects and determines whether it is target or clutter; if it is a target, it outputs its type, size, and location to CPU 2 for symbol generation and the interframe analysis.

The various modules making up bin classification are shown in Figure 5. Each module processes a single bin at a time and passes it on to the succeeding module. Each of these modules is described later in this section.

CPU 1 accesses three memories in the course of its computations. Memory 1 buffers interval data from the interval generation hardware and is used by CPU 1 to develop the object bins. It is also used as a scratch pad in computing object features. Memory 1 is very fast (under 200 nsec access) in order to match the processing speed of CPU 1 and because CPU 1 accesses it quite heavily. Memory 2 stores a digitized video field for the purpose of computing intensity moments. It is slower than Memory 1 since it is not accessed very heavily. CPU 1 only reads from Memory 2; it never writes into it. A third programmable read-only memory (PROM) will store the prototypes for the k-nearest neighbor recognition classifier.

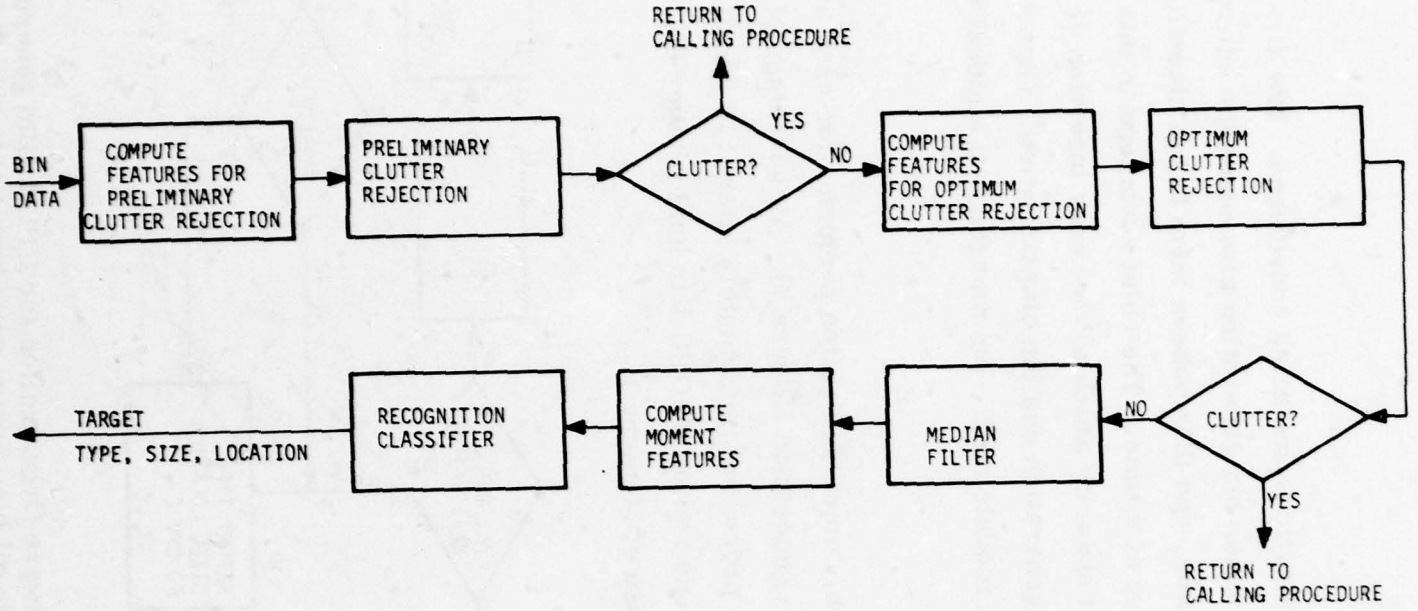


Figure 5. Bin Classification Software

Software Organization

Two alternative organizations are being considered. The first is shown in Figure 6. In this scheme, bin generation processes the entire FLIR image, creates all possible bins, and then passes them to bin classification where they are processed one at a time. The chief advantage to this organization is that it preserves modularity which allows easy insertion of additional modules between bin generation and bin classification. Object data would be available to such a module (e.g., bin merging) on a global (i.e., frame-wide) scale.

In the second approach, bin classification performs for each bin immediately after it is found by bin generation (Figure 7). An advantage of this approach is that, since a bin is processed immediately after it is found, its memory area can be released and reused by CPU 1 to hold another bin. This would allow more efficient use of Memory 1.

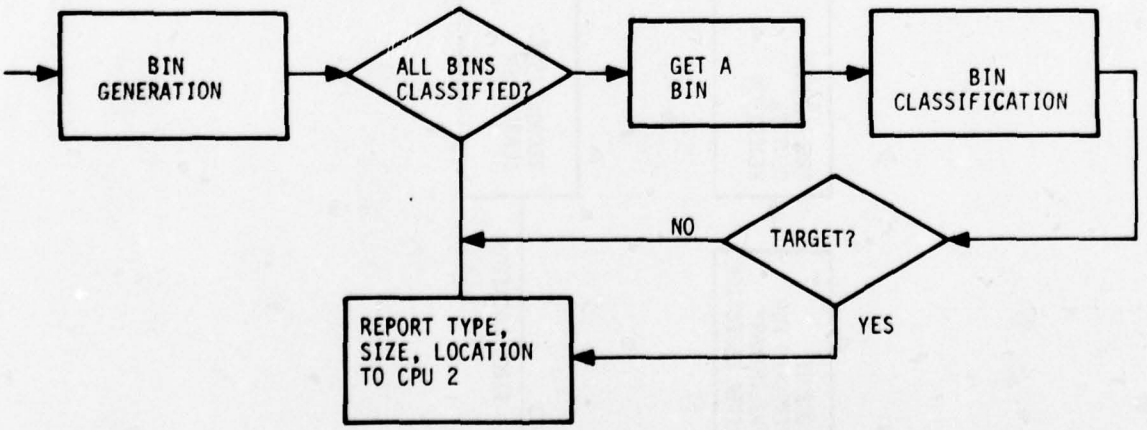


Figure 6. Software Organization for CPU 1 (Bin generation finds all bins before any classification is done.)

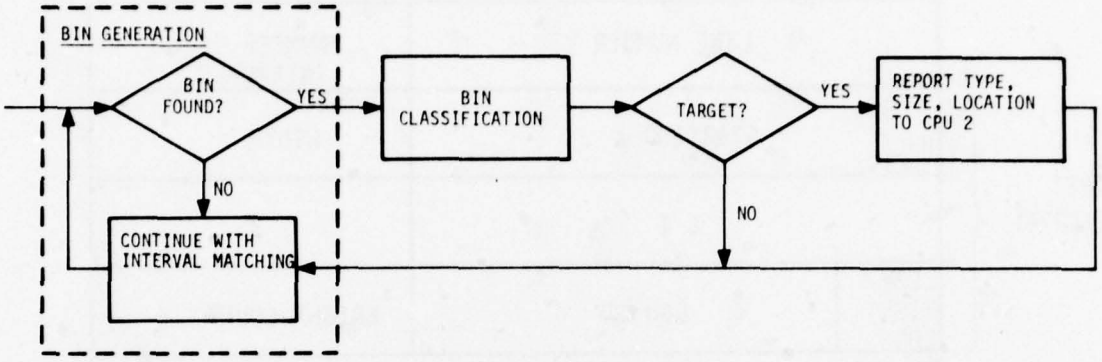


Figure 7. Alternate Software Organization for CPU 1 (Bin generation finds a bin and immediately passes it to bin classification.)

It is likely that the second alternative will be chosen for the implementation because of the memory savings. However, a switch from the second to the first alternative would not be difficult to accomplish.

Software Modules

Bin Generation--

Input: Interval data are dumped to the upper 8 K words of Memory 1 in the format shown in Figure 8.

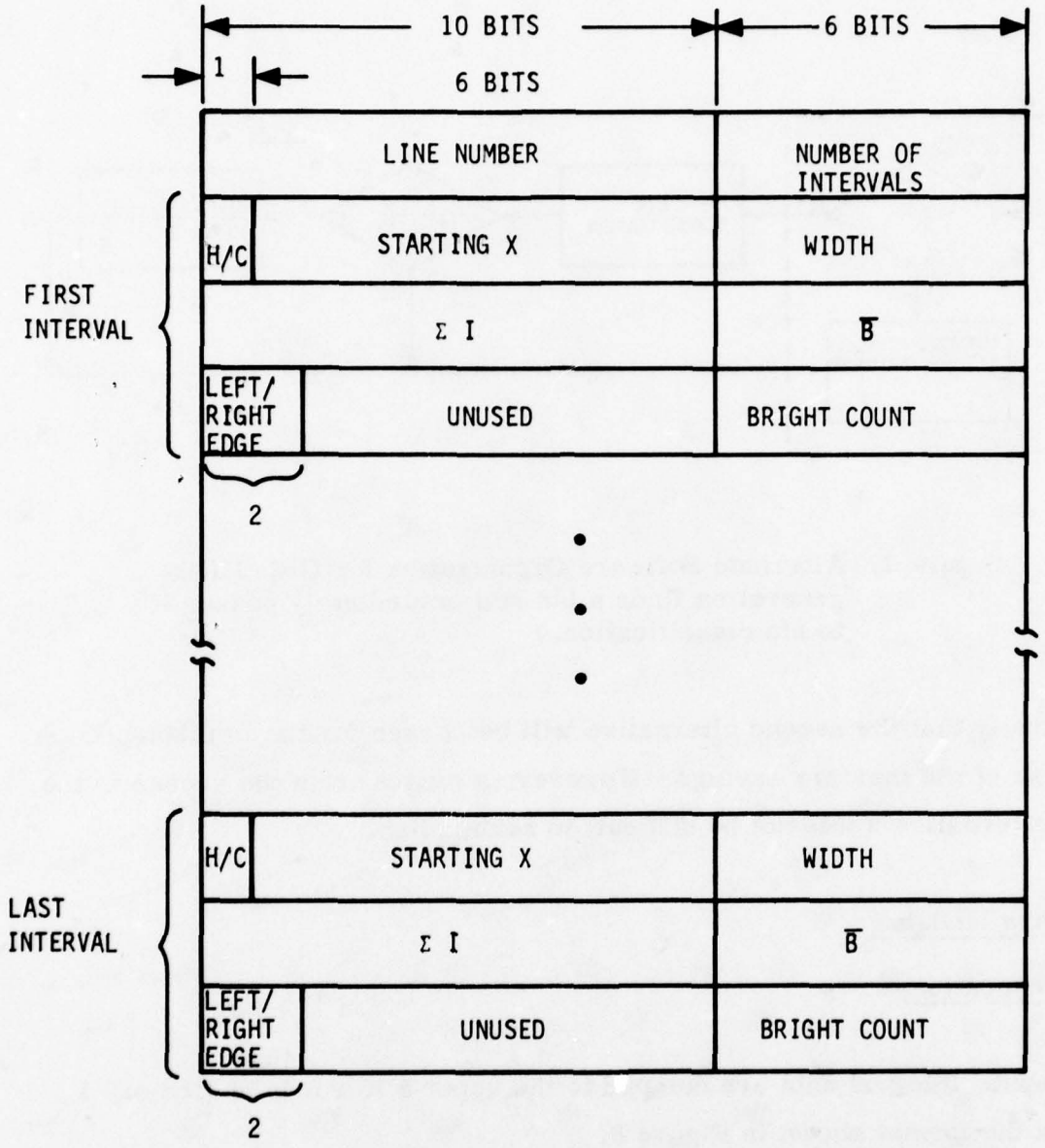


Figure 8. Interval Data Format

Output: Data blocks in Memory 1, each containing interval data, are reorganized into a two-dimensional object or bin. These data blocks have fixed lengths; their format is shown in Table 5.

TABLE 5. BIN FORMAT

Word		Contents
Bookkeeping	1	Address link to next bin
	2	Midpoint (from last interval in bin)
	3	Starting X
	4	X + width
	5	Bin color (H/C)
	6	Line number on which bin starts
	7	Total scan count (N)
	8	*Active scan count*
First interval in bin	9	X
	10	Width
	11, 12	Packed features
	⋮	⋮
Last interval in bin	4N + 5	X
	4N + 6	Width
	4N + 7, 4N + 8	Packed features
	4N + 9	Object features (target contrast, moments, etc.)
	⋮	
	160	

Function: Bin generation (or "bin matching," as it was called in Reference 1) segments the FLIR frame into two-dimensional objects. The concept has been discussed in References 4 and 1. Reference 1 summarizes the implementation.

Figure 9 expands upon the second alternate approach to CPU 1 software organization by highlighting the principal functions of bin generation. Note that two lists of unclosed bins will be maintained: one for hot bins and one for cold bins. This will eliminate hot/cold indicator comparisons between bins and intervals. A list of the locations of available bin areas in Memory 1 will also be maintained. Such a "free list" is necessary because bins close randomly with respect to one another, and we would like to be able to easily find bin memory areas which can be reused.

The hot and cold bin lists will be maintained as linked lists in ascending order according to the value of the midpoint of the last interval associated with each bin. Bin generation will first try to establish overlap between a bin and interval and, once overlap is established, will check for midpoint correspondence. A subsequent PATS quarterly report will contain a detailed description of the bin matching algorithm being implemented.

Preliminary Clutter Screening and Clutter Feature Computations--

Input: Object bins in Memory 1 that are produced by interval matching.

---

[4]D. E. Soland, "PATS Quarterly Progress Report," Contract No. DAAK70-77-C-0248, Honeywell Systems and Research Center, Minneapolis, Minnesota, January 15, 1978.

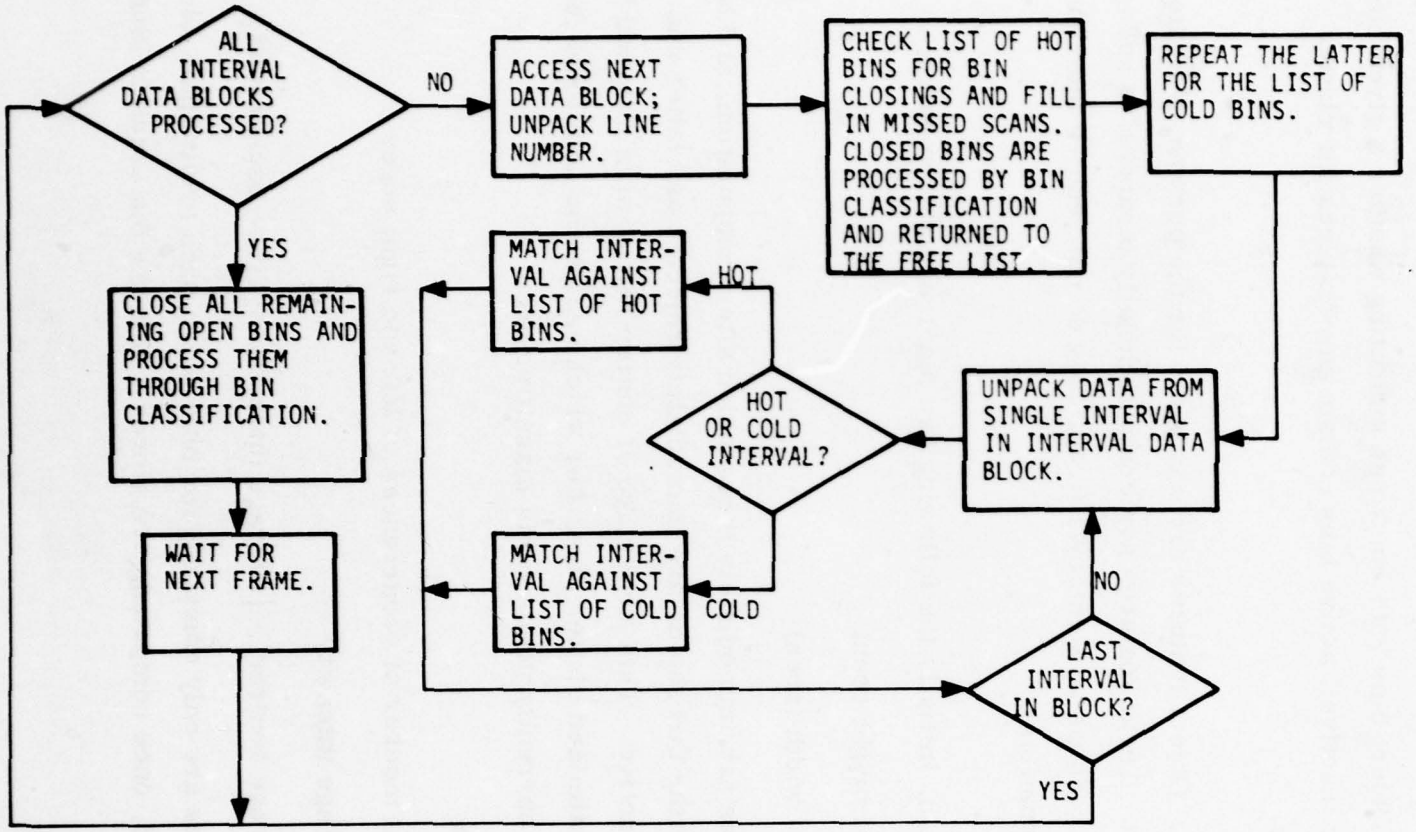


Figure 9. CPU 1 Software Organization

Output: Object bins with markings indicating whether a given bin is active or inactive; active bins contain new features for clutter rejection.

Function: Object features are computed for the purpose of clutter rejection. These features are computationally inexpensive and only involve accesses to Memory 1 (i.e., they do not require individual pixel intensities).

For a given bin, initially the following two features are computed:

- Total bright count
- Total width (area)

These features are immediately thresholded after computation, and bins not satisfying the thresholds are immediately rejected as clutter and marked as inactive. This first stage of clutter rejection is followed by an additional, optimized clutter classifier which uses some additional features. The features currently used by this classifier are:

- Area
- Total number of occurrences of left and right edges
- Average intensity
- Average contrast = |average intensity-average background|

These features are only computed for bins which pass preliminary clutter screening and, once computed, are stored within the bin areas in Memory 1.

Optimized Clutter Classifier--

Input: Object bins in Memory 1 that remain active after preliminary clutter screening. Features are read from each bin as input to the classifier.

Output: Object bins with markings which indicate whether a given bin is active or inactive.

Function: Each bin which passes preliminary clutter screening is processed by the optimum clutter classifier. Bins rejected by this classifier are marked as inactive and are no longer accessed.

The clutter classifier is tree-structured and involves thresholding on the features. The classifier structure is discussed in Section II.

Median Filter--

Input: Object bins which have passed the optimum clutter classifier.

Output: Object bins with modified starting X values and widths.

Functions: Clutter rejection does not use object shape. However, bins which are not rejected as clutter are then processed by the recognition classifier which uses intensity moment features. Intensity moments depend upon object shape, and the median filter smooths the boundaries of a bin using a one-dimensional median filter of width three.

The inputs to the filter are the endpoints of the intervals making up each bin. A separate filtering operation is done on the left-hand and right-hand edges of each object. Each triplet of endpoints on a given edge is sorted, and the middle endpoint of the triplet is assigned the middle value from the sort. Note, however, that this filter is nonrecursive; only original endpoints are input to the filter, not filtered ones. All the intervals within each bin will be processed and, where necessary, starting X values and widths will be updated.

Moment Feature Computations--

Input: Median filtered object bins in Memory 1 which remain active after processing by both preliminary clutter screening and the optimized clutter classifier.

Output: Active object bins, each with four additional moment features.

Function: The intensity moment features μ_{11}, μ_{02}, μ_{20}, μ_{21}, and μ_{03} are computed for the remaining active bins. Their definitions appear in a previous report.[4] Once computed, these features are stored in the bin areas.

Recognition Classifier--

Input: Object bins in Memory 1 remaining active after processing by the clutter classifier. Moment features are read from these bins as input to the recognition classifier.

Output: Object classifications, together with object sizes and locations. These data are stored in an area of Memory 1 for direct memory access (DMA) transfer to CPU 2.

Function: The recognition classifier puts each active object bin into one of five target categories using only the moment features for that object and stores that classification in Memory 1 together with the object size and location (location information is derived from data in the bin). After all bins have been processed by this classifier, CPU 2 is interrupted by CPU 1 and all the classification information is transferred to CPU 2 memory via DMA. Processing then transfers to CPU 2. Current plans have CPU 1 idle during the remainder of its 0.1 second processing frame, but this will probably change after a detailed timing analysis of the PATS implementation software is complete.

The recognition classifier is of the k-nearest neighbor variety and is described in detail in Section II. Timing considerations for this classifier are also discussed there, and the issue of Euclidean versus city block metrics is raised. It turns out that, for the PATS implementation, the Euclidean metric can be calculated just as quickly as the city block metric owing to the presence of the fast (115 nsec) multiplier/accumulator. The microcode for computing the squared Euclidean distance

$$d_E^2(\bar{X}, \bar{Y}) = \sum_{i=1}^{4} (x_i - y_i)^2$$

between two four-tuples $\bar{X} = (x_1, x_2, x_3, x_4)$ and $\bar{Y} = (y_1, y_2, y_3, y_4)$ will be similar to the following:

Instruction Number	Function
1	Clear accumulator register in multiplier.
2	Fetch x_1 from Memory 1; put in R1 (R1 = register 1 on 2903 ALU).
3	Fetch y_1 from Memory 1; form R1 - y_1; store result in R2 and in Y latch on multiplier/accumulator.
4	Store R2 in X latch on multiplier.
5	Enable multiply/accumulate function. Fetch x_2 from Memory 1; put in R1.
6	Fetch y_2 from Memory 1; form R1 - y_2; store result in R2 and in Y latch on multiplier/accumulator.
7	Store R2 in X latch on multiplier.
8, 9, 10	Repeat instructions 5, 6, 7 for x_3 and y_3.
11, 12, 13	Repeat instructions 5, 6, 7 for x_4 and y_4.
14	Enable multiply/accumulate function.
15	Fetch final result from accumulator register on multiplier.

15 instructions x 200 nsec/instruction = 3 μsec/distance calculation

On the other hand, the microcode for computing the city block distance

$$d_c(X, Y) = \sum_{i=1}^{4} |x_i - y_i|$$

would be as follows:

Instruction Number	Function
1	Clear R2 in 2903 ALU.
2	Fetch x_1 from Memory 1 and store in R1.
3	Fetch y_1 from Memory 1; form R1 - y_1; store result in R1.
4	If R1 > 0, go to instruction 6.
5	Negate R1.
6	R2 ← R2 + R1
7 8 9 10 11	Repeat instructions 2, 3, 4, 5, 6 for x_2 and y_2.
12 13 14 15 16	Repeat instructions 2, 3, 4, 5, 6 for x_3 and y_3.
17 18 19 20 21	Repeat instructions 2, 3, 4, 5, 6 for x_4 and y_4.

21 instructions x 200 nsec/instruction = 4.2 μsec/distance calculation

Thus, because of the multiplier/accumulator, Euclidean distance calculations are roughly 1 μsec faster than city block calculations.

The prototypes for the k-nearest neighbor classifier will be stored in fast 4 K x 16-bit PROMs (allowing 1000 prototypes, four features per prototype, and one 16-bit word per feature). This PROM will be in the CPU 1 address space along with Memory 1.

CPU 2 SOFTWARE

The definitions of the CPU 2 software modules have not changed since the previous quarterly report. This software is diagrammed in Figure 10 and summarized in this section.

The software is divided into operational and diagnostic sections. The operational software is used both when the PATS system is running normally and when it is being tested or trained. The diagnostic software, however, is used only when the system is being tested or trained.

Operational Software

Interframe Analysis--

Input: Object classifications and locations in Memory 1 produced by CPU 1 from a single frame of data.

Output: A cumulative classification for each object based on the interframe analysis.

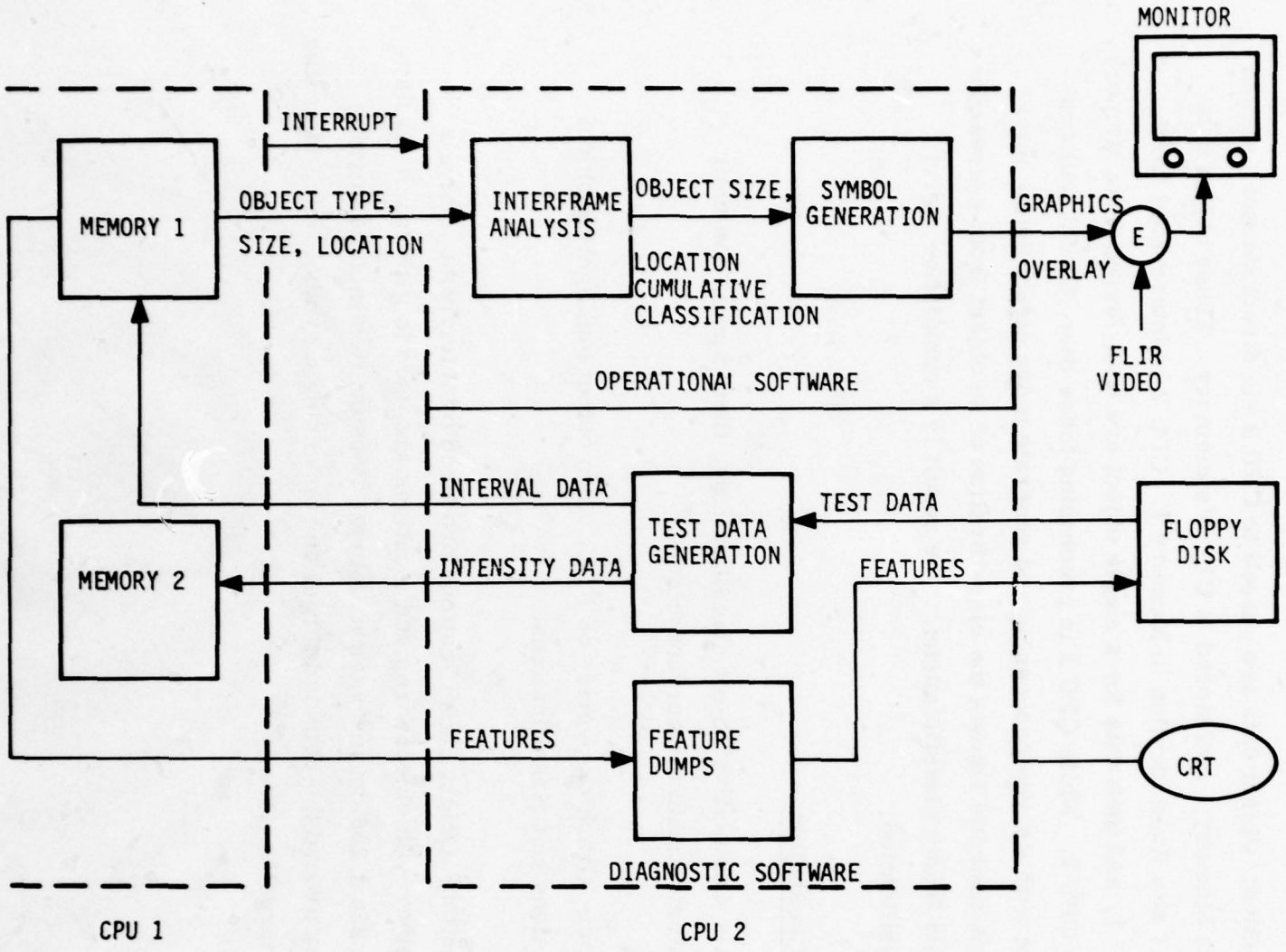


Figure 10. CPU 2 Software

Function: Object data are passed to CPU 2 via direct memory access from Memory 1 and stored in CPU 2's memory. These object data start at a fixed location in Memory 1. CPU 2 is interrupted from CPU 1, and then data for a single object are transferred from Memory 1 to CPU 2. While CPU 2 is processing these data, CPU 1 will continue with its own processing and generate more object data. Interframe analysis adjusts the classification of an object from successive single frame classifications. The result is a cumulative object classification.

Symbol Generation--

Input: Cumulative object classifications, their sizes, and their locations within the current frame.

Output: Graphics overlay on FLIR video output highlighting object locations and classifications.

Function: Object classifications produced by interframe analysis together with object sizes and locations are used to generate a display in a 256 x 256 dot graphics memory. Possibilities include drawing a box around the detected target and appending a symbol which indicates the target's class.

Diagnostic Software

Test Data Generation--This module will load test data into Memory 1 and/or Memory 2 in CPU 1 for the purpose of testing the implemented PATS algorithms on known data. The test data will be generated either internally or will be loaded from tape or floppy disk.

Feature Dumps--This software will interact with CPU 1 to accomplish dumps of interval and object features from Memory 1 onto tape, floppy disk, line printer, or CRT. These dumps will be used to check out CPU 1 software and to do off-line classifier training. This software will also dump information to help check out the CPU 2 operational software.

SECTION IV

HARDWARE DESIGN

The PATS hardware design tasks have been broken down into the following subparts:

1. Image enhancement
2. Edge signal
3. Bright signal
4. Interval generation
5. CPU 1
6. Memory 2 (intensity information)
7. CPU 2
8. Symbol generation
9. Sync and timing

The functions and requirements of items two through nine are covered in the hardware specification. Item one has been covered in a previous report.

This section discusses those designs which are nearly complete and being submitted for build. For each part a detailed block diagram is presented along with a discussion of how the circuit works and major parts used in the design. Specific values of components are not provided for resistors and capacitors.

EDGE SIGNAL

The edge signal derivation is shown in Figure 11. The video output from the FLIR or from the image enhancement is used for the video input. The edge operator is two-dimensional in that it uses three scan lines of data for determining an edge. The video data are delayed by two one-line delay

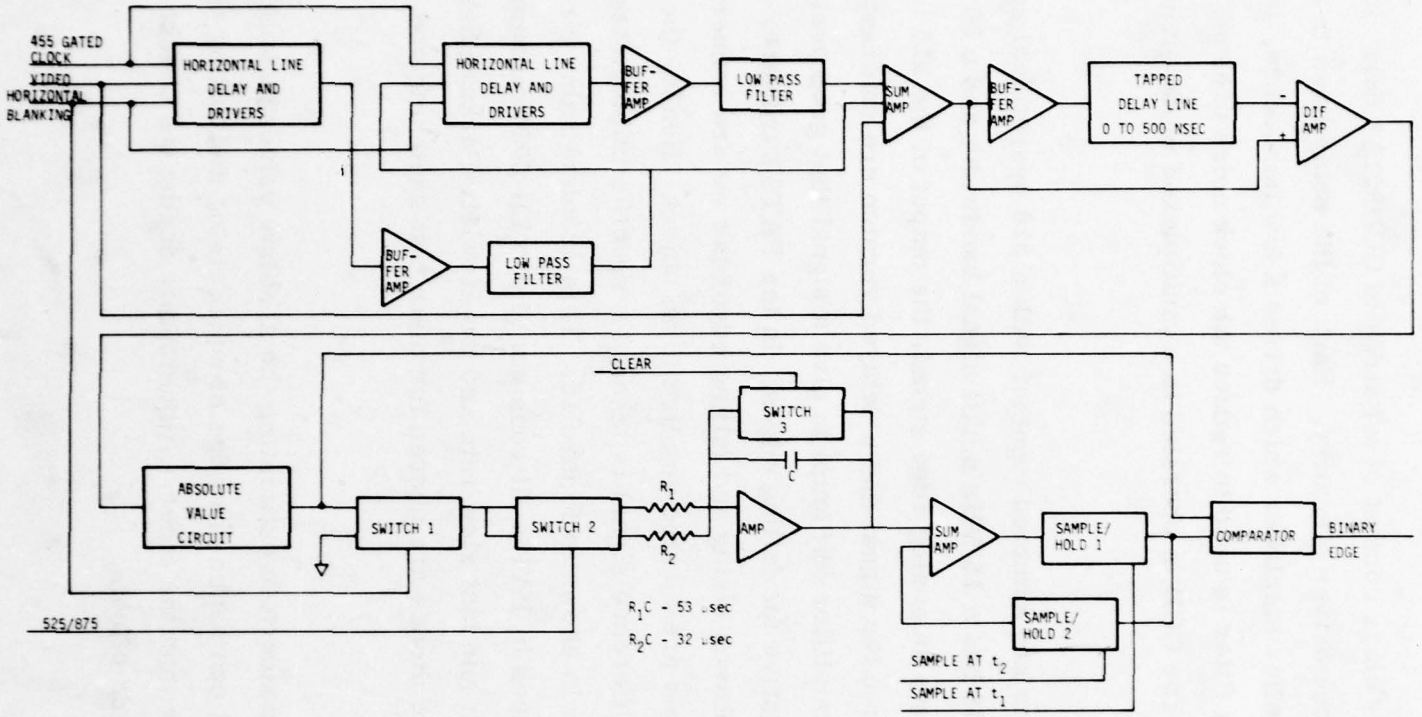


Figure 11. Edge and Edge Threshold Hardware Implementation Block Diagram

lines. These delays consist of a Fairchild CCD321A3 delay line and the associated clock driver circuitry. Each of the delay lines is followed by a unity gain buffer amplifier which drives a low impedance, low pass filter. This low pass filter is used to reduce the clock noise present on the output of the CCD. The CCD is operated in a multiplexed mode giving 910 samples/line.

The three lines are summed together with a 318 operational amplifier (op amp). The 318 has a 15 MHz small signal bandwidth and a 50 V/μsec slew rate. To create an analog edge signal, the output of the 318 is delayed a few pixels; then the signal and a delayed version are subtracted in the differencing amplifier (dif amp) to give a signal that goes positive for rising edges and negative for falling edges. In the PATS hardware, we do not distinguish between rising and falling edges but we are concerned with where the edges are located relative to an object. Hence, the absolute value of the difference signal is taken. A specific implementation of the absolute value is shown in Figure 12. This absolute value circuit is used in several places in PATS. It consists of two LH 0024 op amps which have high bandwidth and fast slew rate and two Hewlett-Packard Schottky barrier diodes. These diodes are chosen for their fast switching and low turn-on voltage.

The primary concern in estimating the absolute values is uniformity. That is, a dc offset can cause one edge of equal rising and falling edges to end up slightly larger than the other. Adjustments during the checkout phase will eliminate any dc offsets.

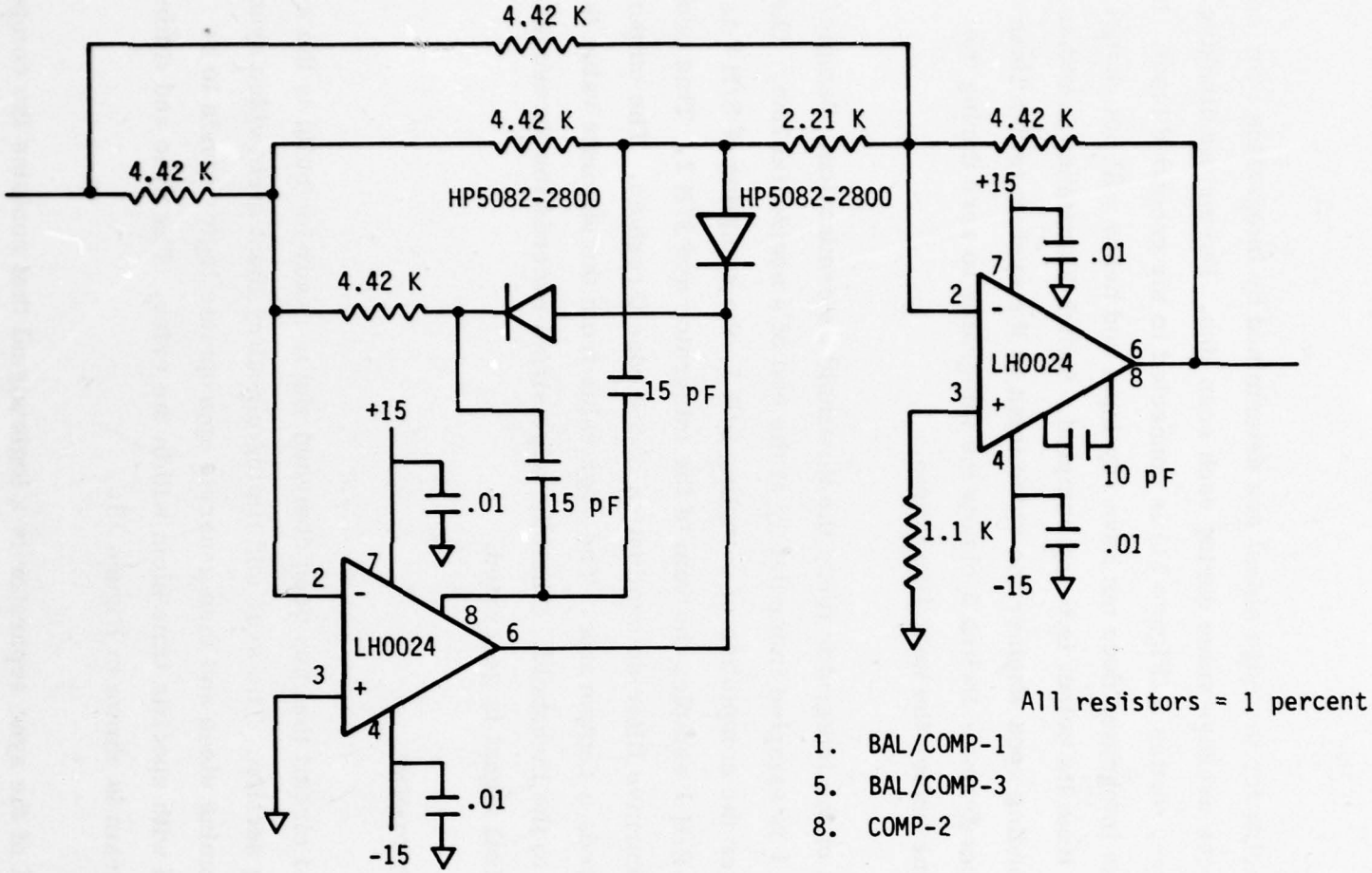


Figure 12. Absolute Value Circuit

The thresholds for the edge signal are determined by integrating over each line to get the average value during each scan line. During the blanking or retrace time, switch 1 (Figure 11) is connected to the grounded input. In this way the integrator does not have any input and hence will not change during the time its output is being sampled. Switch 2 switches in either R_1 or R_2 depending upon whether the video input to the system is 875 lines or 525 lines per frame. Switch 3 clears the integrator to zero during the retrace time before the next line starts.

The output of the integrator feeds the threshold determination. Sample/hold (S/H) 1 is sampled immediately at the end of a horizontal line. Then shortly after the completion of sampling S/H 1, the sampling of S/H 2 is initiated. S/H 1 samples the sum of the integrator and S/H 2. This provides a recursive filter determination of the edge threshold. The output of S/H 1 feeds a comparator. The edge value from the absolute value is compared to the threshold. When the edge signal exceeds the threshold, then a logical signal is generated.

SYNC AND TIMING

The second circuit that has been designed and is ready for build is the sync and timing section. The sync and timing circuitry must strip video sync out of the incoming video and then generate appropriate logic signals to be associated with specific time slots within the video. The sync and timing block diagram is shown in Figure 13.

The output of the sync separator is a logic signal that contains the composite sync (CSYNCD). It is slightly delayed relative to the incoming video. The composite sync signal feeds both a horizontal and a vertical reset logic

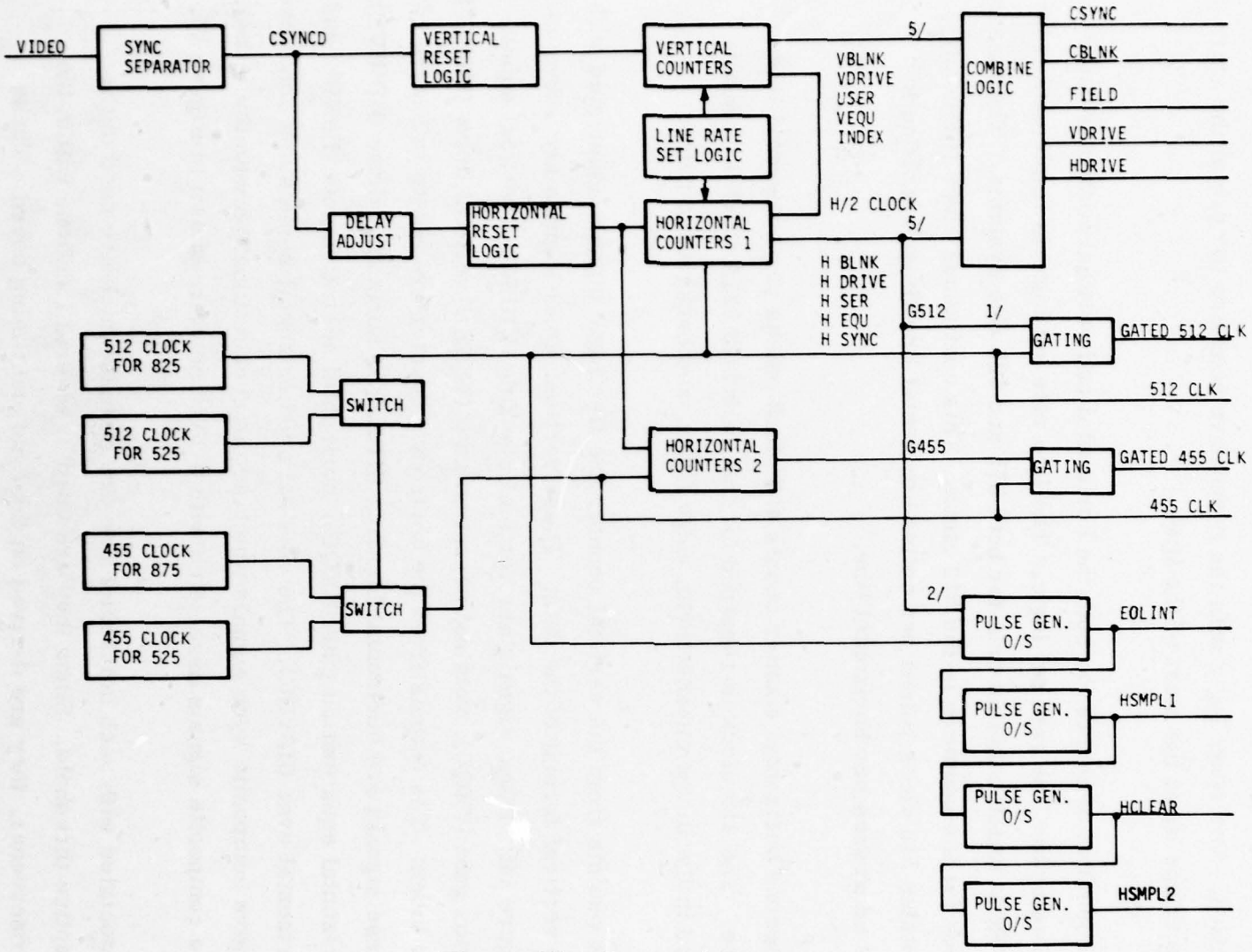


Figure 13. Sync and Timing

section. The reset logic sets the respective counters to zero at the start of a frame or at the start of the line.

The number of counts for both the horizontal and vertical timing is determined by the line rate set logic. The line rate set logic is implemented by a PROM which contains data for both 875 and 525 line systems. The master clock for the counters is the 512 clock. This continuous clock frequency contains 512 clock pulses per active horizontal line time, resulting in 512 addresses per horizontal line.

A second horizontal counter counts 455 clock pulses per horizontal line time. The 455 clock is required for the Fairchild 321A CCD line delays used in the image enhancement, edge filter, and background filter.

The outputs from the vertical counter are five logic signals associated with the vertical format of the data. These vertical timer signals are shown in Figure 14. These signals are vertical blanking (VBLNK), vertical equalization gate (VEQU), vertical serration gate (VSER), vertical drive (VDRIVE), and index. The outputs from the horizontal counter are shown in Figure 15. These signals are horizontal blanking (HBLNK), horizontal drive (HDRIVE), horizontal equalization gate (HEQU), horizontal serration gate (HSER), and horizontal sync (HSYNC). The vertical and horizontal signals are combined to give composite sync signals which are in synchronization with the video. The composite signals associated with a TV frame are shown in Figure 16.

Associated with each horizontal line are signals which are used in the adaptive threshold. Since they are used in several locations within the autoscreener, they are derived on the sync and timing board. These

signals, shown in Figure 15, are the end of line interrupt (EOLINT) used to interrupt CPU 1 to allow first level features to transfer to Memory 1, horizontal sample/hold 1 (HSMPL1) used to sample the line integrator output, horizontal clear (HCLEAR) used to clear the line integrator, and horizontal sample/hold 2 (HSMPL2) used to determine the filter threshold values.

The sync and timing board also outputs a continuous clock along with the gated version. The sync and timing board has independent oscillators for the 875 and 525 line systems.

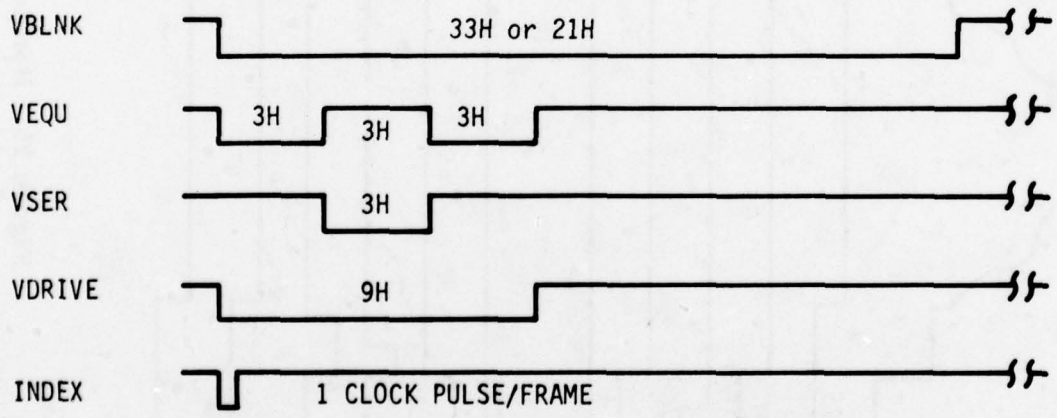


Figure 14. Vertical Timing

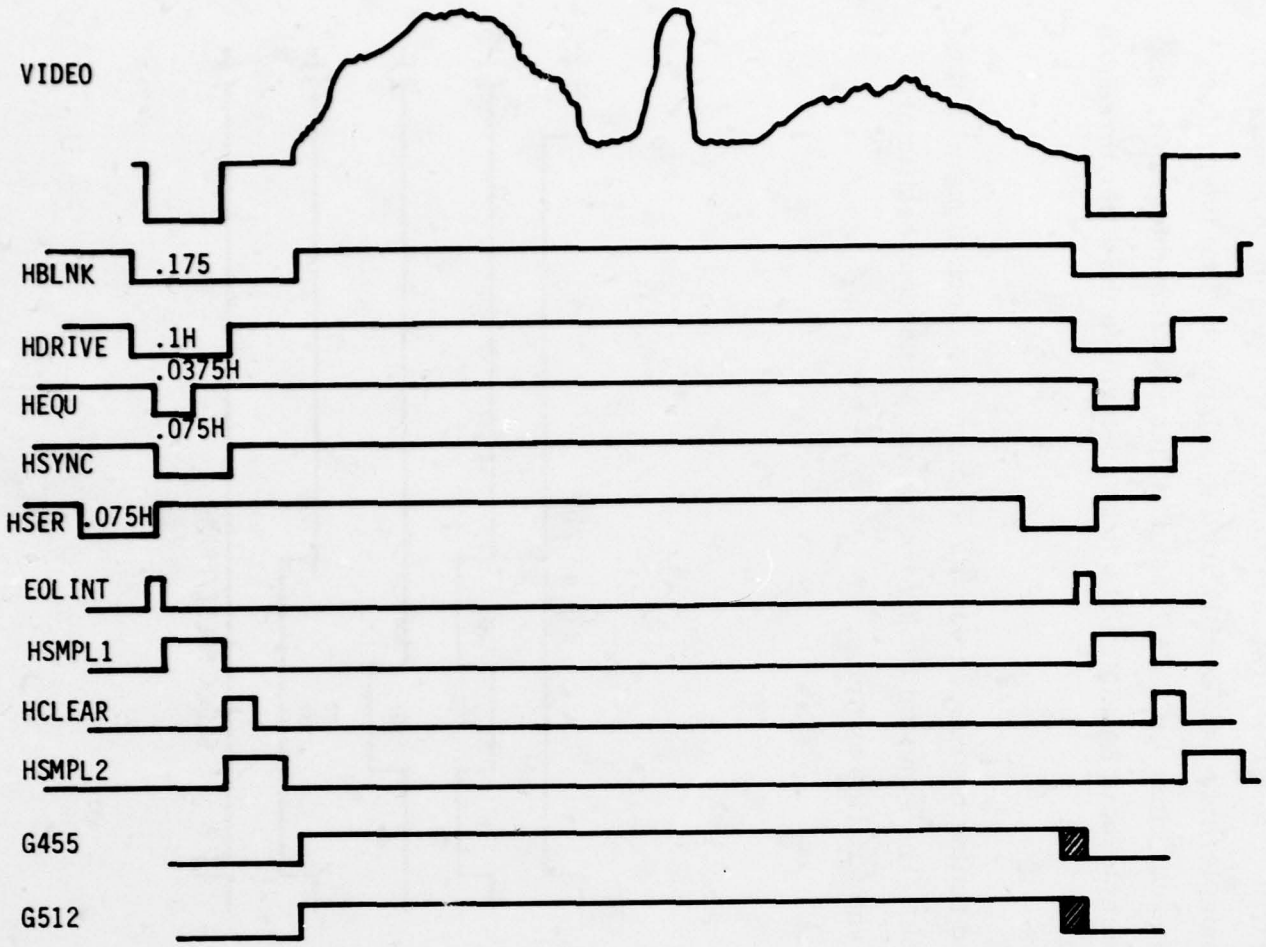


Figure 15. Horizontal Scan Line Timing

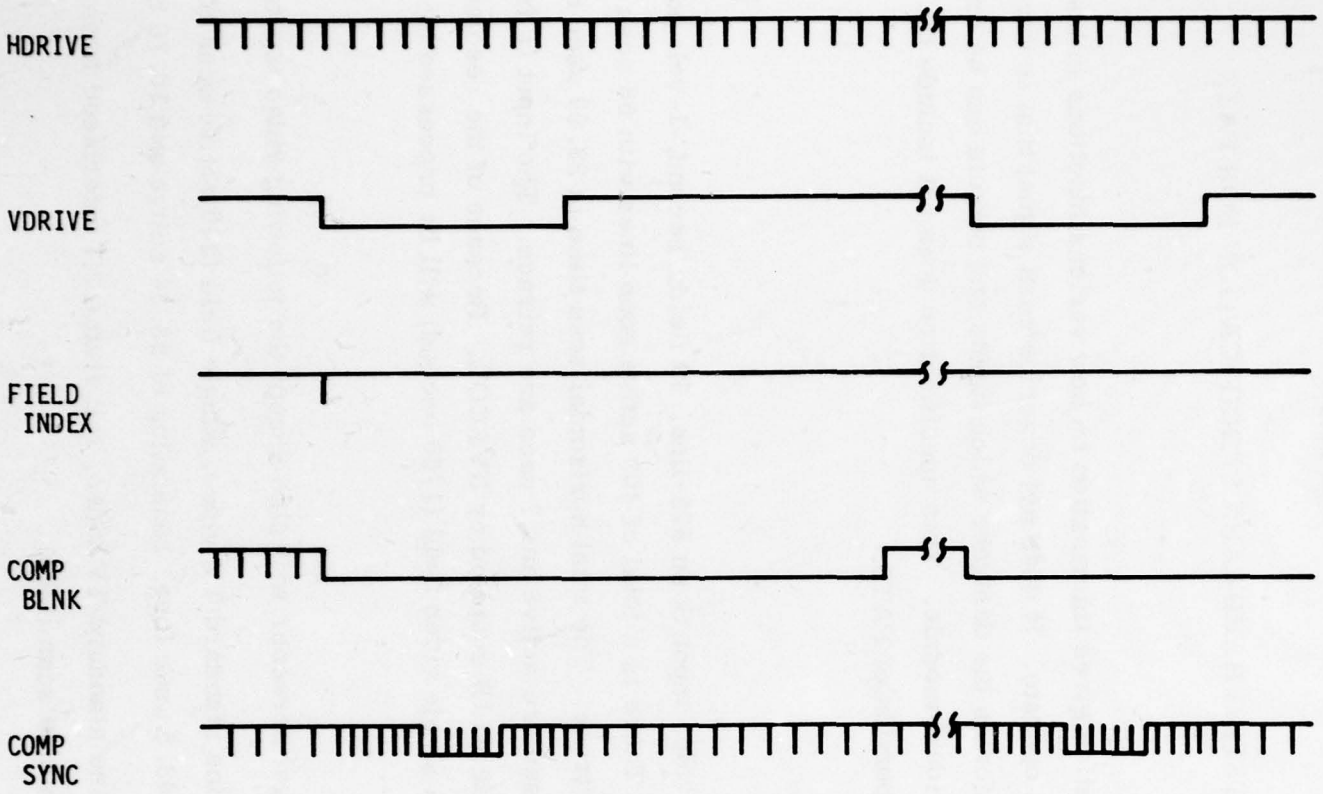


Figure 16. Composite Frame Sync (shown for 525)

SECTION V

SYSTEM HARDWARE SPECIFICATION FOR PATS

This specification gives information on how various functions in the PATS hardware will operate. It does not describe each signal that is derived on a board but informs the designer which inputs and outputs can be expected for each individual module. This specification does not include the image enhancement portion of PATS.

VIDEO INPUT

The standard video input is an 875-line, 60 fields/second, 1-volt peak video signal. There is a total of 809 active scan lines with 66 scan lines for vertical retrace. The total horizontal scan time is 38.09 μsec of which 31.09 μsec are active and 7 μsec are retrace. The input is from a common module FLIR supplied by NV&EOL. Because of the resolution of the FLIR, a single video field (1/60 second) will be processed by the hardware.

The PATS target screener will also accept the following video inputs:

- 525-line standard TV video, single field (240 active scan lines, each 63.5 μsec long, consisting of 53.34 active and 10.16 blanking)
- 525-line standard TV video, full frame 2:1 interlaced format (480 active scan lines)

PATS HARDWARE

The PATS hardware consists of seven basic functions or modules shown in Figure 17. These modules will be broken up into several submodules and discussed in greater detail later. The main modules to be discussed are:

1. Autothreshold--Accepts a video input and automatically determines the edge and bright logic signals.

2. Interval Generation--Logically combines and validates the edge and bright signal generated by the autothreshold and generates an interval across a potential target.

3. First Level Features--Generates the features that are directly transferred to a processing system.

4. Computer System 1--The main processing computer consisting of a 16-bit, 2901-based CPU, 24 K x 16 memory, and a multiplier/accumulator. Its function is to do bin or object matching, generation of additional features, and recognition.

5. Computer System 2--Provides for diagnostic capability as well as interframe analysis and symbol generation software.

6. Intensity Data--Provides for real-time digitization of the video data at 512 samples/horizontal line. Its maximum size is 512 x 512 x 8 bits.

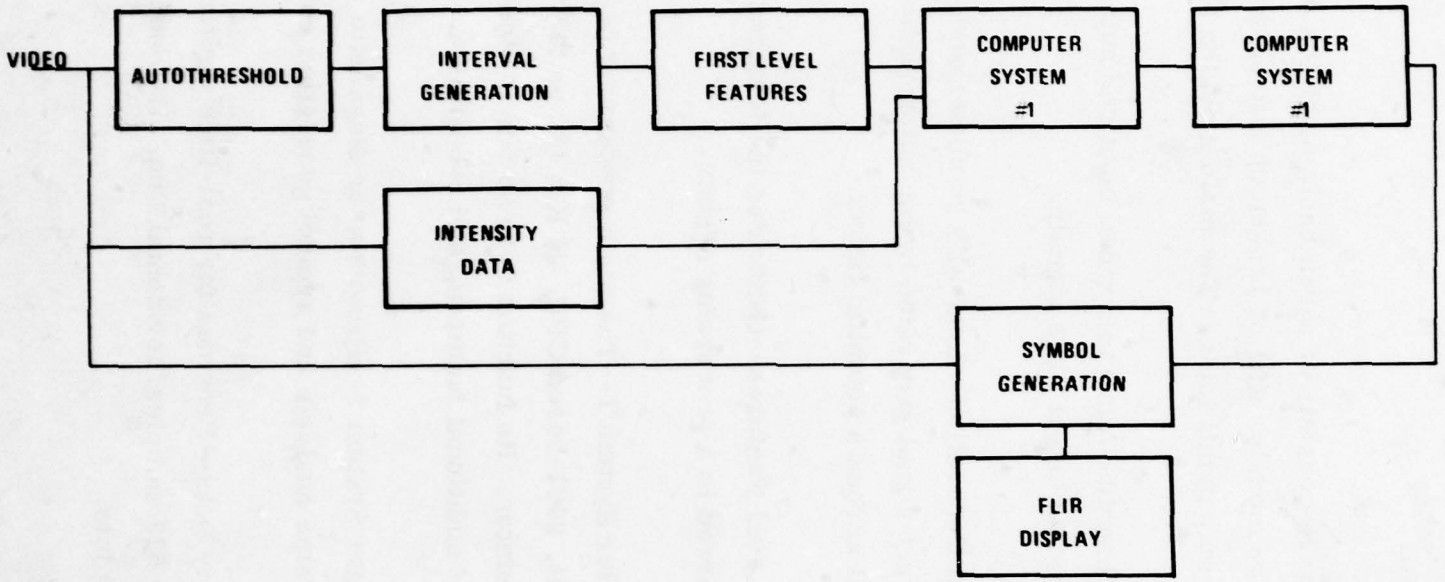


Figure 17. PATS Functional Units

7. Symbol Generation--A memory used for the generation of a specific symbol associated with a target. At some specified location, the symbol is displayed on the FLIR display.

For each of these seven functions, the type of inputs expected to be available, the outputs expected, signals that have to be derived on the board, and the technology to be used will be described.

AUTOTHRESHOLD FUNCTION

The autothreshold section (Figure 18) is primarily an analog processing section. This section derives the logic signals associated with an edge, and hot and cold objects. This is done by comparing an analog signal with an adaptive threshold. The result of the comparison gives a logic signal stating whether the data are above or below the threshold.

The autothreshold consists of two separate parts: the edge signal derivation and the hot or cold signal derivation. The specific requirements for each of these sections follow.

The edge signal is based upon the following algorithm:

$$\begin{aligned} \text{Edge} &= I(n+1,k) + I(n,k) + 2I(n-1,k) \\ &\quad - I(n+1,k-1) + 2I(n,k-1) - I(n-1,k-1) \end{aligned}$$

where

n = scan line number

k = position in scan line

1 = delay time

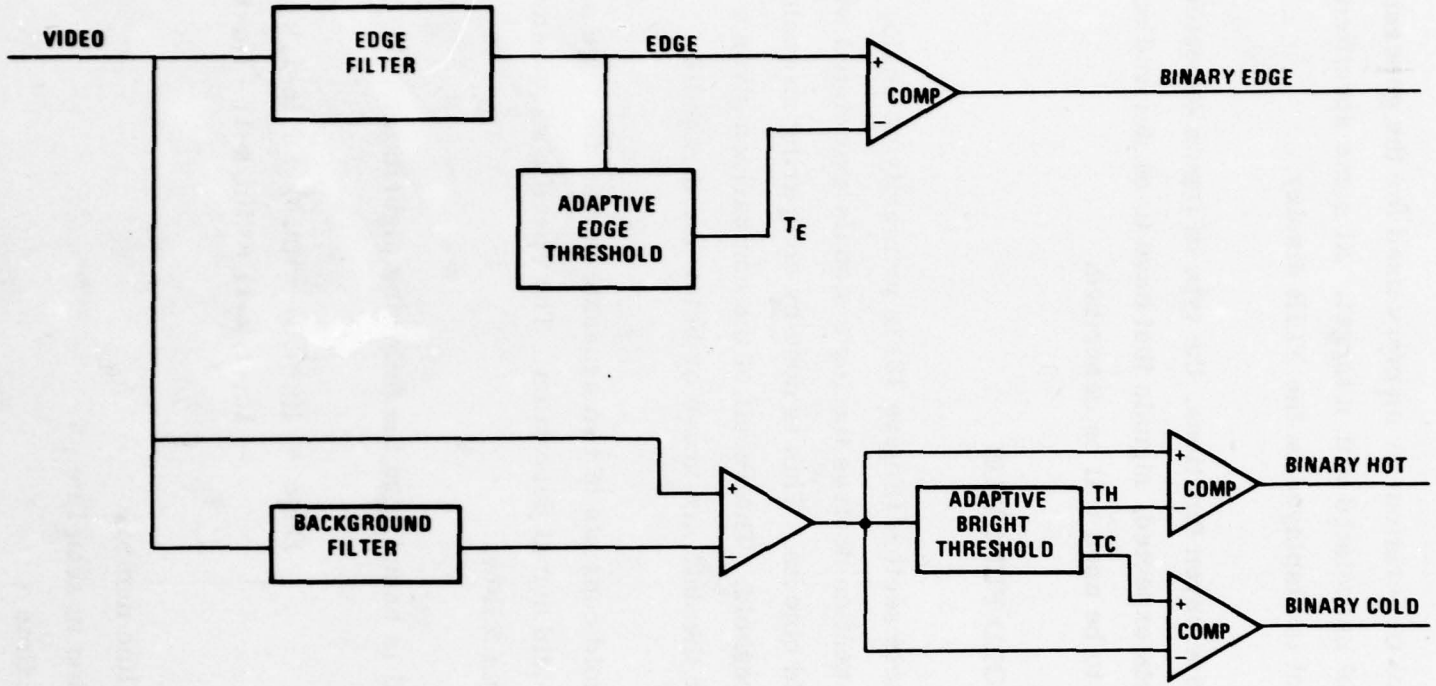


Figure 18. Autothreshold Structure

The implementation is shown in Figure 19. The edge threshold is T_E:

$$T(E) = K(E) * VAR(N-1)$$

$$VAR(N) = \alpha * VAR(N-1) + (1-\alpha) * 1/SL \Sigma |e|$$

where

$K(E)$ = a constant

α = recursive filter constant

SL = scan line time

$|e|$ = edge signal absolute value

The implementation for the edge threshold is shown in Figure 20. Table 6 gives expected available inputs, the type of output expected, some of the signals that will have to be derived on the board, and the type of technology or special parts that will have to be used.

The second portion of the autothreshold is the adaptive bright signal generation (Figure 21). This primarily consists of a background filter and an adaptive threshold.

The background estimator is a two-dimensional recursive low pass filter. This filter, shown in Figure 22, consists of a switch which controls the input, a low pass filter whose RC time constant is a few pixels, and the recursive filter whose parameter β determines the number of scan lines it takes to build up to the background level.

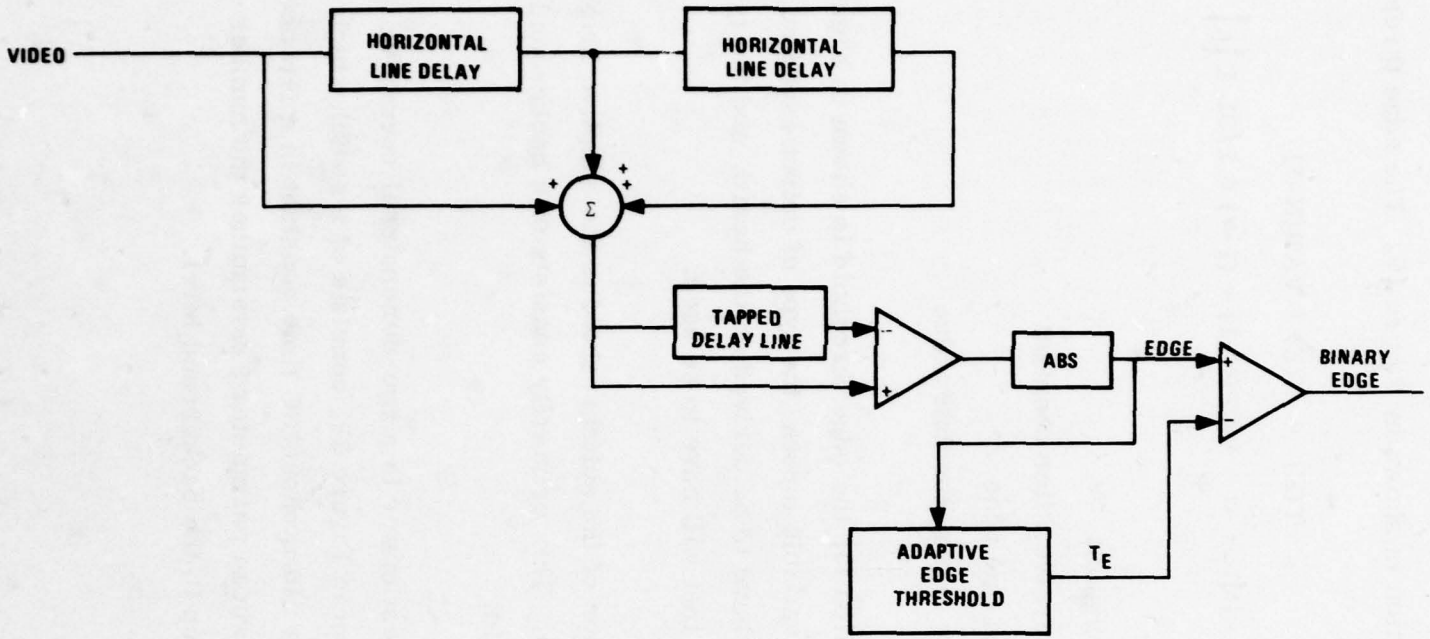


Figure 19. Edge Signal Derivation

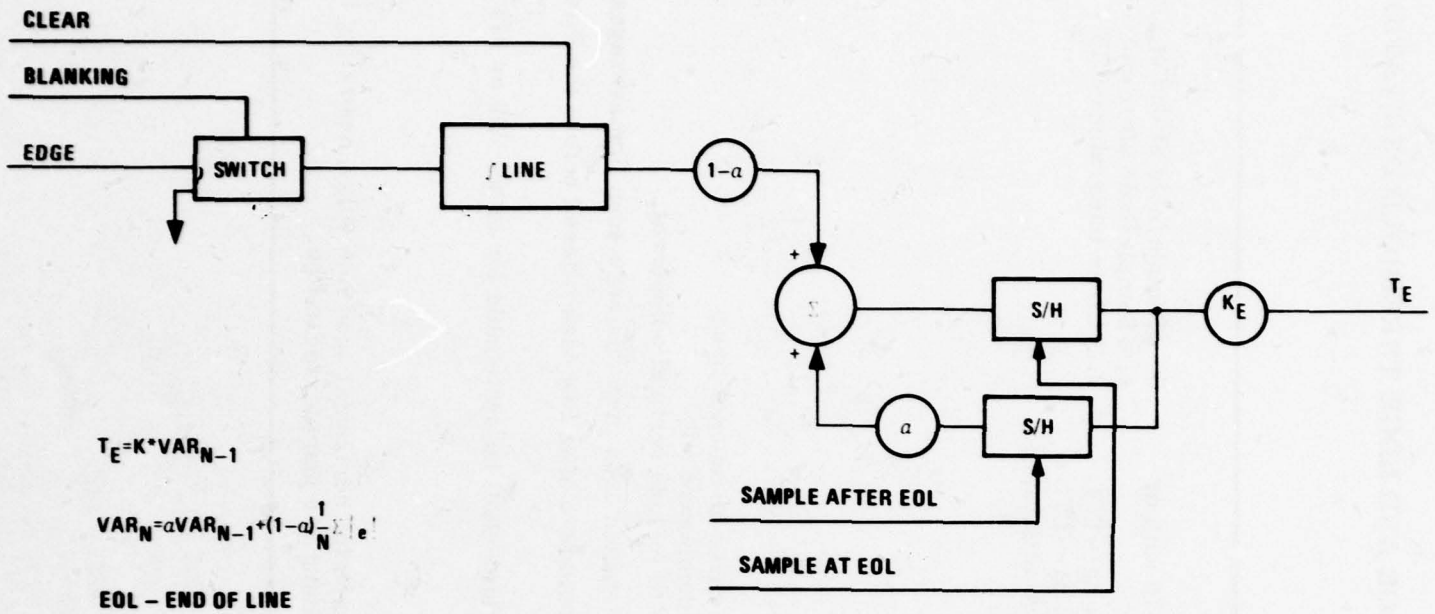


Figure 20. Adaptive Edge Threshold

TABLE 6. EDGE AND EDGE THRESHOLD DERIVATION

Inputs:

1. Video--525 or 875 with or without sync
2. Composite blanking--TTL
3. Horizontal blanking--TTL
4. Clock--455/horizontal line ungated--TTL
5. Sample/hold at end of line--TTL
6. Sample/hold after end of line--TTL
7. Clear integrator--TTL

Outputs:

TTL--edge

Derived in Hardware:

1. *Edge*--
   - Based upon three horizontal lines
   - Horizontal component only
   - Expandable to include vertical component
2. Edge threshold--based upon previous edge scan line averages.
3. Analog edge should be 5 MHz low pass filtered before the threshold is derived.
4. Scan line integrator shall be switchable for either 525 or 875 line time.

Technology:

CCD321A3 will be used for horizontal scan line delay, operating in multiplexed mode giving 910 samples/scan line.

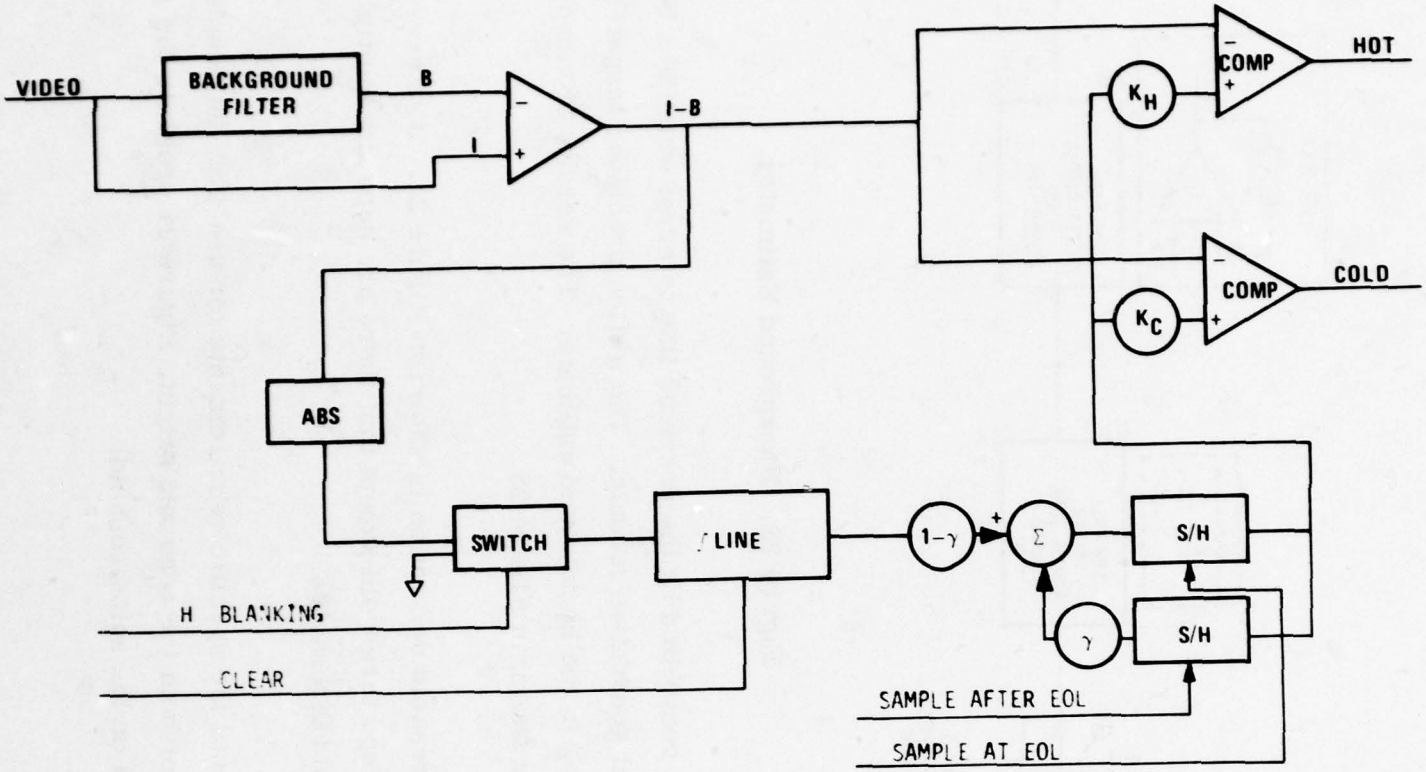


Figure 21. Adaptive Bright Threshold

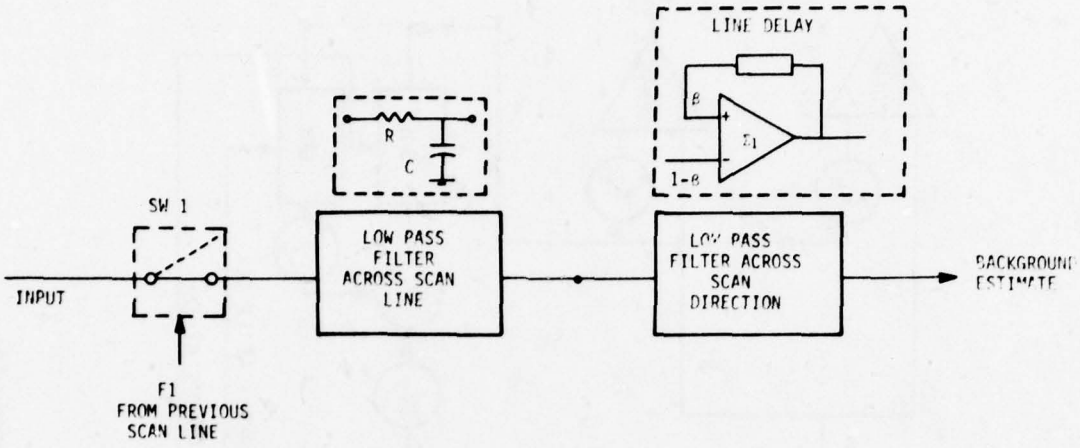


Figure 22. Background Estimator

The switch is controlled by the previous line interval which has been stored by the interval generation module. The switch precludes target-like data from appearing in the background estimate. The results of video without the switch are shown in Figure 23.

The bright threshold derivation is shown in Figure 21. It is very nearly the same as the edge threshold except that there are separate multipliers for the hot and cold thresholds.

Table 7 presents the specific requirements for use with the adaptive bright threshold. For both the edge and bright, Figure 24 presents the relative timing for use on the autothreshold.

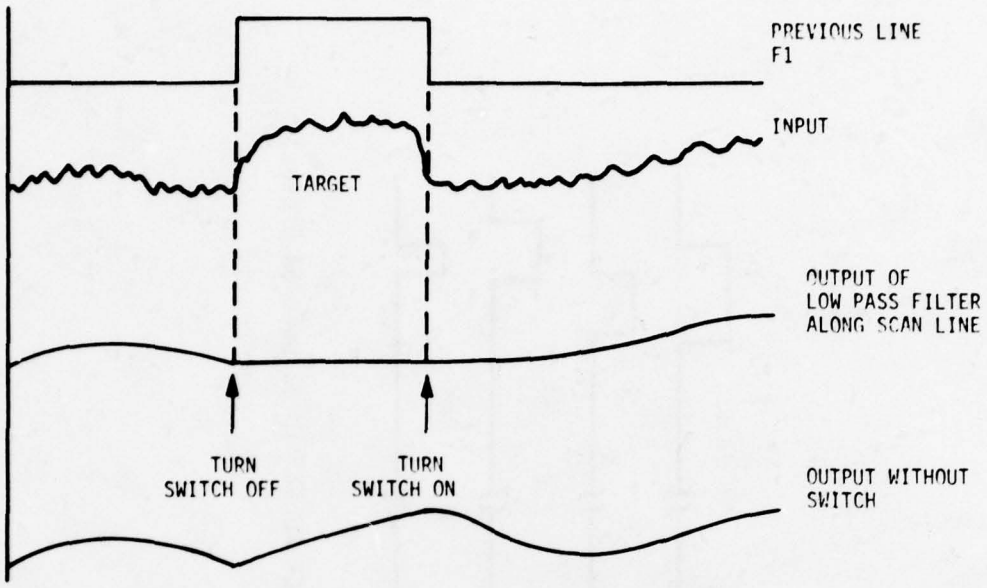


Figure 23. Background Filter and Switch Rationale

TABLE 7. ADAPTIVE BRIGHT DERIVATION

Input:

- 1 volt pp. video 525 line or 875
- Composite blanking TTL
- Horizontal blanking TTL
- Clock 455/horizontal line time
- Previous line interval
- Sample/hold at end of line--TTL
- Sample/hold after end of line--TTL
- Clear integrator--TTL

Output:

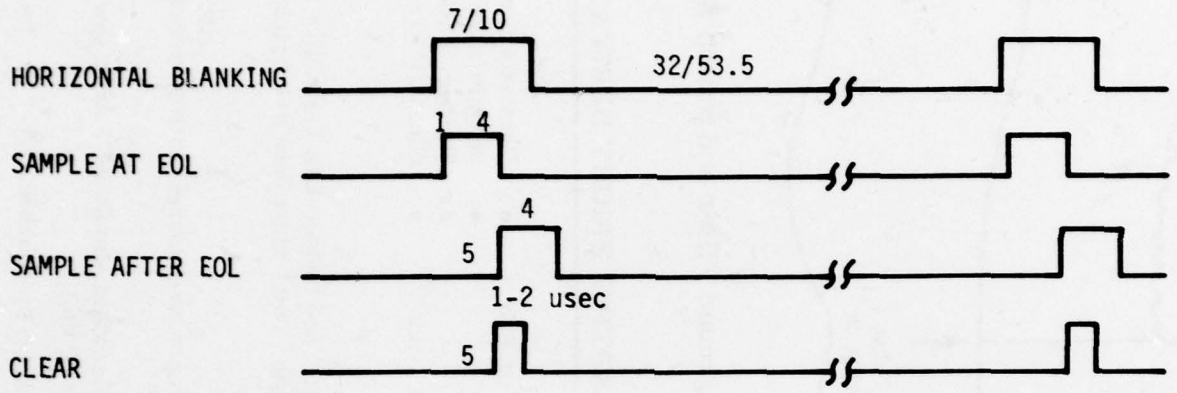
- Binary hot--equal to logic "one" when data are above the hot threshold
- Binary cold--equal to logic "one" when data are below cold threshold

Derived in Hardware:

1. Bright threshold--based upon previous scan line averages of intensity-background.
2. Intensity-background analog data should be 5 MHz low pass filtered before the threshold is derived.
3. Scan line integration shall be switchable.

Technology:

CCD321A3 will be used as part of the recursive filter.



CLOCK 1 CONTINUOUS 14.21875 MHz/8.5047 MHz (455 CLOCK PULSES IN 32 usec/53.5 usec)

Figure 24. Relative Scan Line Timing for Autothreshold

INTERVAL GENERATION

The interval generation (Figure 25) is the logical combining of edge and bright information that determines the regions of possible targets. It is in synchronization with a master digital clock.

In the start/stop criteria (Figure 26), a potentially bright interval is filtered by an M or N criteria (M = 4; N = 5). If there are M of N brights, the interval is turned on. Note that delay lines must be provided to give proper phasing of the edge and bright signals. Part of the phasing difference comes from the analog processing and part is inherent in the start/stop criteria.

The edge data must also be filtered by a set of M of N (M = 3, N = 5) requirements. In the edge criteria, we have a valid edge if any one of the three adjacent lines has a good edge or if three adjacent lines each have an edge associated with three pixels (0, $\pm$1) of the point being considered. This is shown in the bottom half of Figure 26.

The validation criteria requires that an interval be declared valid if any one of the following three conditions occur:

1. An edge occurs at the start of a bright.
2. An edge occurs at the end of a bright.
3. An edge occurs at both the start and end of a bright.

A bit must be provided for conditions 1 and 2. If both conditions 1 and 2 occur, both bits must be set.

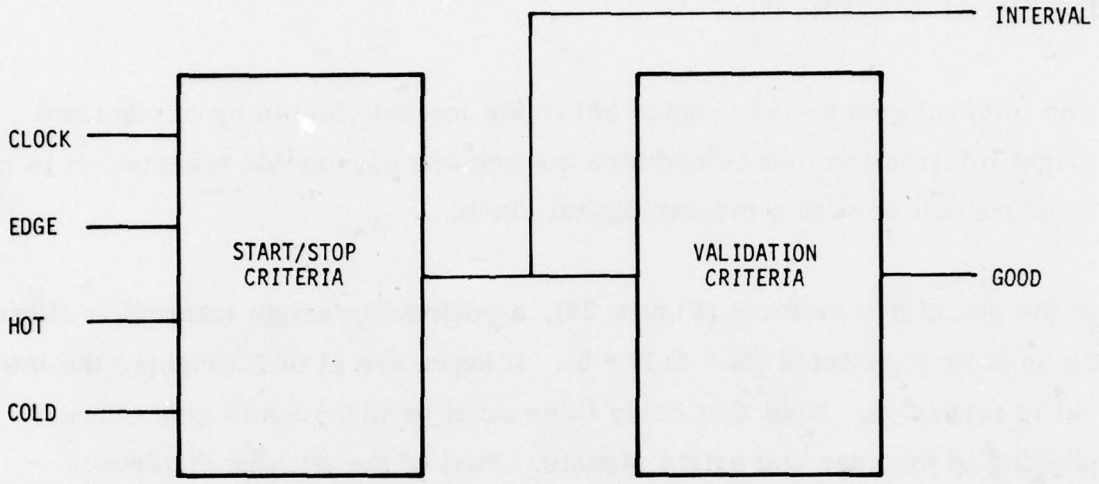


Figure 25. Interval Generation

The interval data that are stored for controlling the switch in the background estimator must only be valid interval data.

An interval will also be declared invalid if the interval width exceeds 32.* A bit will be set indicating that the width has exceeded 32. The width count will then indicate by how much it has exceeded 32 up to a maximum of 32.

This states that we can declare an interval between 32 and 63 as invalid and still have the correct count. However, if the interval exceeds 63, the invalid interval bit must still remain set and the width count shall be set to zero.

*The total width count for a TV line is 512. Thus, the maximum interval is 1/16 of a TV line.

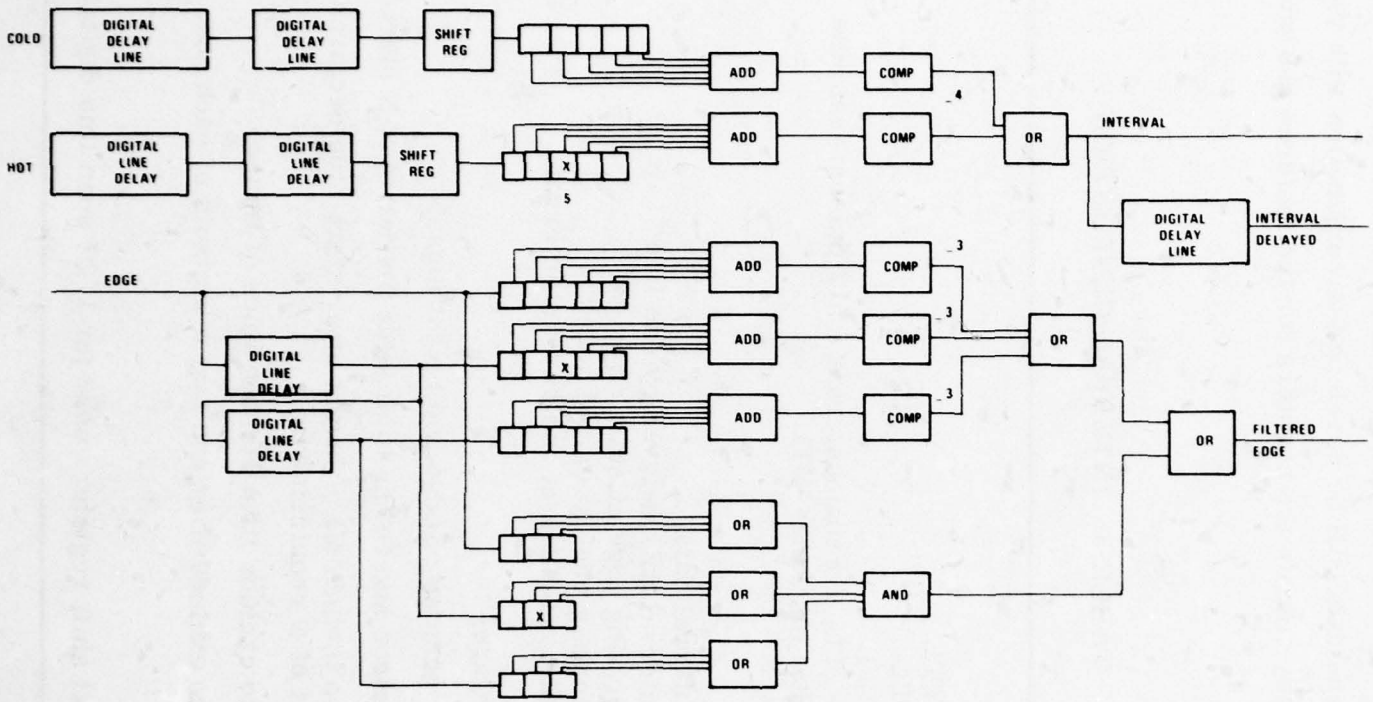


Figure 26. Start/Stop Criteria

Table 8 presents the input and output requirements for the valid interval generation. These data will be developed in conjunction with the first level features since they contain the counters for the features and multiplexers for the data.

TABLE 8. INTERVAL GENERATION

Input:

- Edge--TTL
- Hot--TTL
- Cold--TTL
- Clock--TTL, continuous, rate = 512 pulses/scan line
- Horizontal blanking--TTL

Output:

1. Valid interval data
2. Valid interval data delayed by one line
3. Bit indicating edge at start of bright
4. Bit indicating edge at end of bright
5. Valid interval pulse at end of valid interval

Derived in Hardware:

1. Proper phasing of edge and bright signal
2. Comparison and ORing to provide proper M of N start/stop criteria
3. Delay to include all of bright after start/stop declares that data meet M of N requirements
4. Pulse to exclude first 10 scan lines of image
5. Pulse to exclude first and last few pixels of each scan line

Technology:

TRW 256-bit shift registers used for 1/2H scan line digital delay

FIRST LEVEL FEATURES

Associated with each of the good or valid intervals are several features that contribute to the detection and recognition of an object. These are known as the first level features.

The valid interval pulse must be counted for each scan line. If the number of valid intervals exceeds 16, a bit shall be set and no more data shall be allowed to transfer for that scan line.

Three 16-bit words are generated or available as part of the first level features. With least significant bits first, there are:

Word 1

Width	6
X Position	9
Hot/Cold (H/C)	1

Word 2

Background Estimate	8
Sum of Intensity	8

Word 3

Bright Count	6
Spare	6
Edge at Start	1
Edge at End	1

In Figure 27, the first level feature organization is shown. For word 1, the width count is a count of the width of an interval. It must be counted even though it may later be declared an invalid interval because there are no edges. The position is the true position of the start position of the interval in the scan line. The H/C bit indicates whether the data are above or below the background.

Word 2 is described in the intensity data description. The data must be latched at the end of an interval, however, so more data can be accumulated.

Word 3 contains the count of the number of brights during an interval. It will always be less than or equal to the width count. Also in word 3 is the edge count, which is the number of edges during the interval and the associated bits indicating the location of edges relative to bright.

Table 9 presents information on the first level feature generation. Note also that this requires interfacing with the interval generation, CPU 1, and the intensity data.

INTENSITY DATA

The intensity data consist of two separate digitizations. These are the background feature (Figure 28) and the video intensity (Figure 29). The background is an 8-bit digitization of the analog background estimate taken at the start of an interval. At the start of an interval, a fast sample/hold is initiated and then converted within 325 nsec. This becomes the background estimate which goes to the first level features.

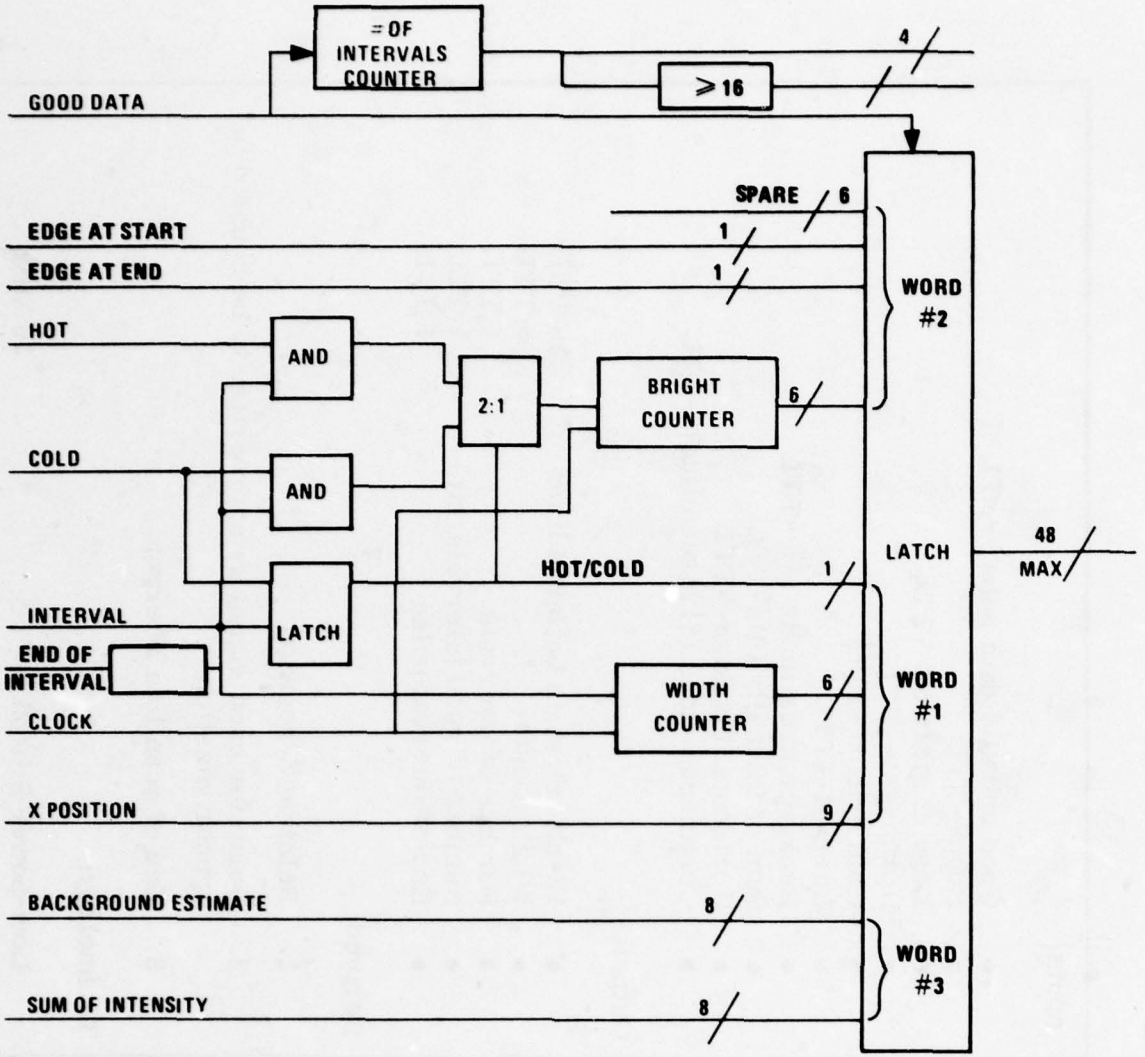


Figure 27. First Level Features

TABLE 9. FIRST LEVEL FEATURES

Inputs:

- Good or valid data pulse--TTL
- Edge--TTL
- Edge at start bit--TTL
- Hot bright--TTL
- Cold bright--TTL
- Interval--TTL
- Background estimate (8)--TTL
- Sum of intensity (8)--TTL
- Horizontal blanking--TTL
- Continuous clock (512/scan line)--TTL

Outputs:

• 16-bit tri-state buffered lines	16 TTL
• Width counter	6 TTL
• Number of intervals	4 TTL
• Number of good intervals/bit	1 TTL
• End of interval pulse	1 TTL

Derived:

1. Relative X position
2. Delay for each signal as appropriate to be counted in synchronization
3. End of scan line interrupt

Technology:

Low-power Schottky	16 MHz

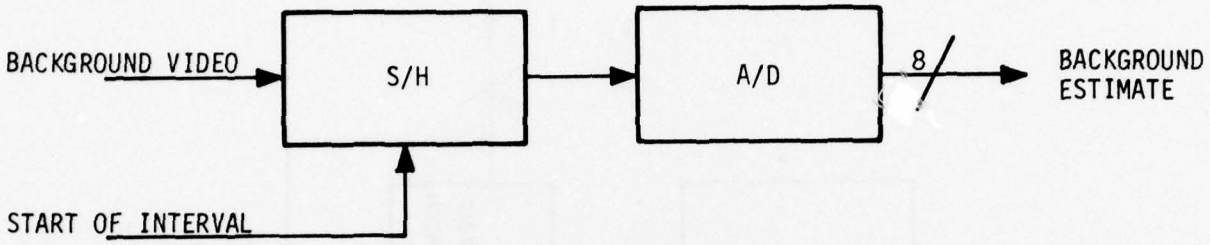


Figure 28. Background Feature

The video intensity data consist of a high speed A/D conversion (62 nsec for 875 line), a high speed 12-bit adder, and an intensity memory. The A/D is on for the entire field and all data are digitized and stored. There are a total of 512 samples per scan line with 6 bits of intensity. The A/D converter is an 8-bit converter but only the 6 MSB are used in the application. Since the data maximum value is 2^6 and the maximum interval length is 2^5, 11 bits in the adder would be sufficient. It is anticipated that 4-bit summing modules would be used and hence the 12 bits. At the end of the interval, the 12-bit sum is latched and transferred to the first level features. The latch is cleared after the data have been transferred.

The intensity data to be stored in Memory 2 are changing approximately every 65 nsec. In order for memory to accept this fast rate, a scheme must be developed which would allow the memory to be low power and yet

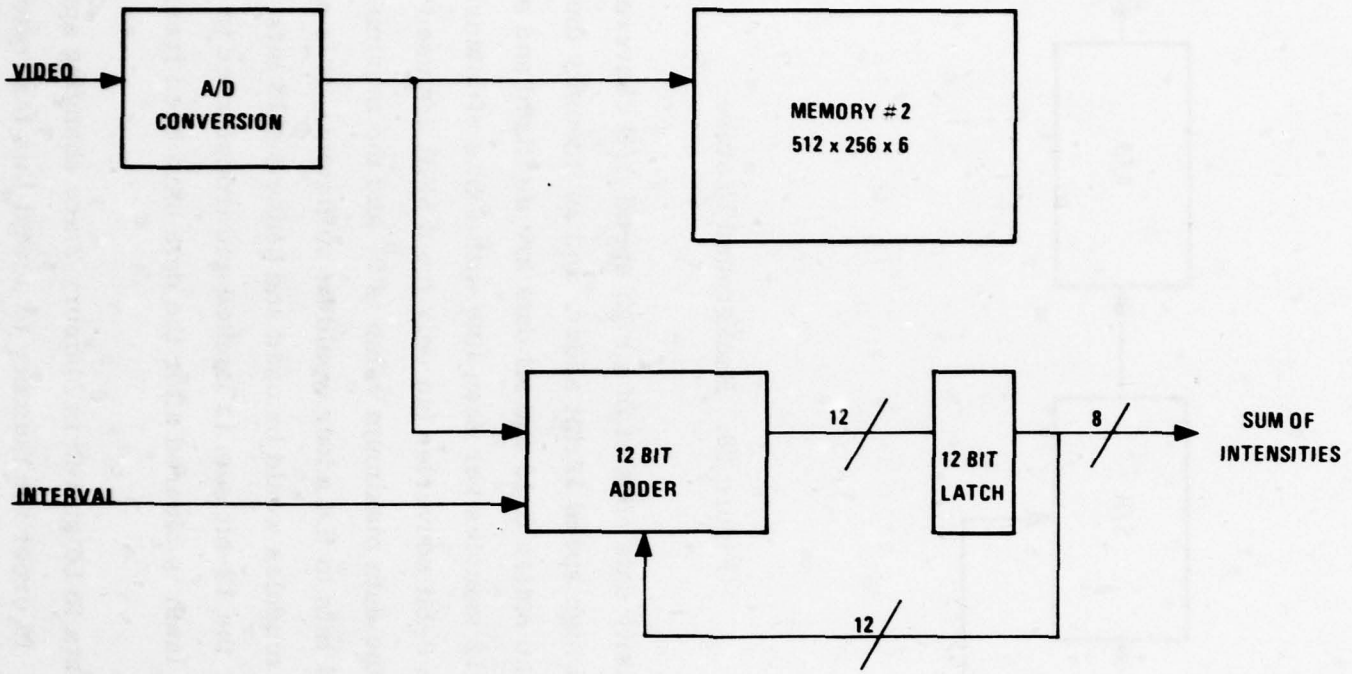


Figure 29. Intensity Feature

be randomly accessible. Such a scheme is shown in Figure 30. A single bit plane is implemented with shift registers on the input and output. Average random access time from CPU 1 should be around 200 nsec. The scheme presented allows for eight values to be available with a cycle time of about 375 nsec. One only has to give a starting address and then an address increment eight larger to get the next eight values, etc.

Table 10 gives the background requirements, and Table 11 presents the video intensity requirements. Memory 2 requirements are shown in Table 12. All interact to provide the intensity data.

TABLE 10. INTENSITY DATA--BACKGROUND

Inputs:

- Interval--TTL
- Continuous clock--TTL
- Background estimate-analog--0 to 1 V

Output:

- 8-bit background estimate

Derived within:

1. Sample/Hold at start of interval
2. Conversion complete

Technology:

High speed A/D--300 nsec

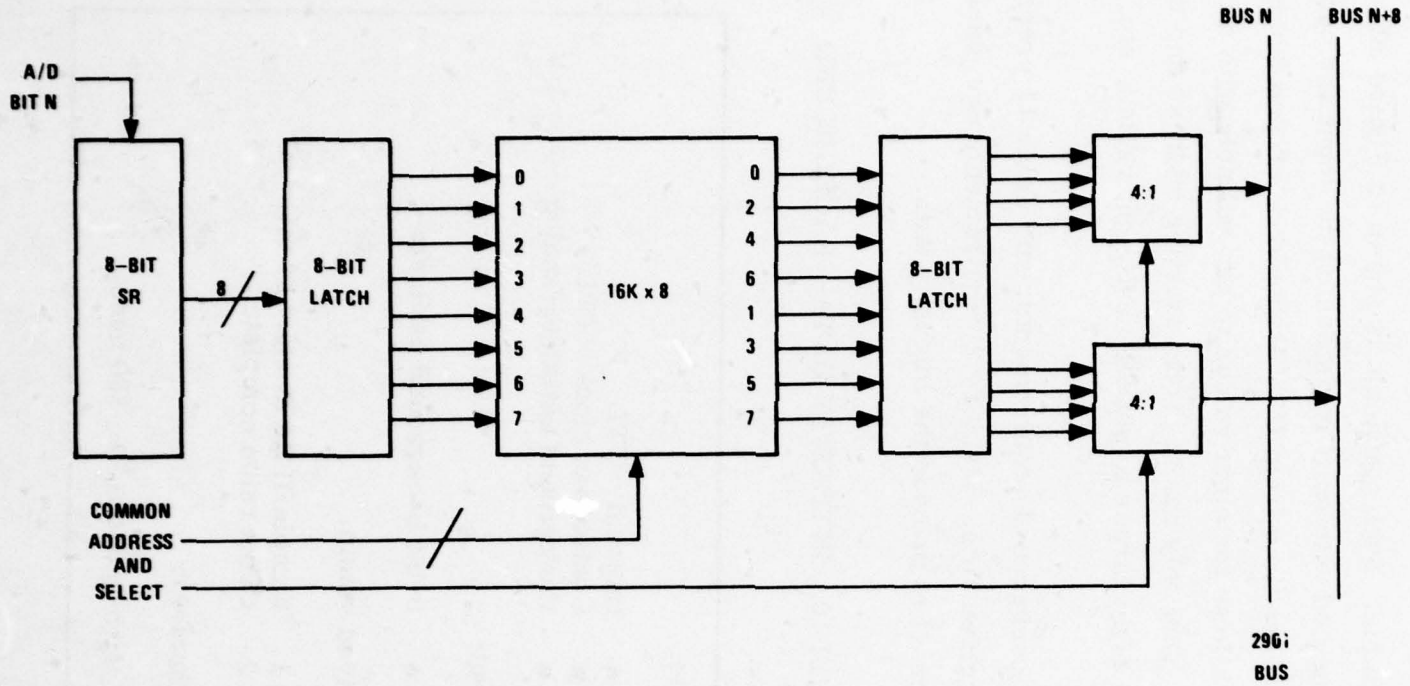


Figure 30. Video Field Memory Bit Plane

TABLE 11. INTENSITY DATA--A/D AND SUM OF INTENSITY

Input:

- TV video--Analog 0 to 1 V
- Horizontal blanking--TTL
- Composite blanking--TTL
- Continuous clock--TTL

Output:

- 6-bit intensity data--TTL
- 12-bit sum of intensity over interval--TTL (8 bits to FLF)

Derived:

- Convert data at clock rate pulse
- End of summation

Technology:

- A/D--TRW 8-bit video converter
- Low-power Schottky

TABLE 12. INTENSITY DATA--MEMORY 2

Input:

- 6-bit intensity data--TTL
- Continuous clock--TTL
- Horizontal blanking--TTL

Output:

- 6-bit, 512 x 256 video data stored in image format

Derived within:

- X address
- Y address
- Interface to CPU 1
- Interface to CPU 2

} Do with respective CPU

Technology:

- Fujitsu Dynamic RAMs, 16 K x 1

Notes:

Expandable to 512 x 512 x 8 bits

COMPUTER SYSTEM 1

Computer system 1 shown in Figure 31 must accept the inputs from the first level features' portion at the end of each interval. It must also access the intensity memory (Memory 2), be able to use a parallel I/O multiplier/accumulator, and interface to CPU 2.

At the end of each interval, an end of interval interrupt will occur which will tell the hardware that is part of CPU 1 to load 3- to 16-bit words into FIFO 1. The FIFO will have approximately 325 nsec to accept the three words before a new set of data might be available. At the end of each scan line, the number of intervals for that scan line will be available so that DMA transfer of the data can occur. Data from FIFO 1 to FIFO 2 will occur and then data will be transferred to Memory 1. The CPU will wait for the DMA transfer to occur. The Memory 1 interface is shown in Figure 32.

Memory 2 will be accessed typically after all the data in Memory 1 have been preprocessed. Generally Memory 2 will be used as input to the multiplier/accumulator for moment calculations.

The interface between CPU 1 and CPU 2 will handle DMA access to Memory 1 and Memory 2 and might possibly handle the microprogram memory transfer during development of the software.

Table 13 gives some of the requirements of CPU 1 and its associated memories.

AD-A060 850 HONEYWELL INC MINNEAPOLIS MINN SYSTEMS AND RESEARCH --ETC F/G 17/5
PROTOTYPE AUTOMATIC TARGET SCREENER.(U)
SEP 78 D E SOLAND, P M NARENDRA, R C FITCH DAAK70-77-C-0248
UNCLASSIFIED 78SRC54-3 NL

2 OF 2
AD
A060850

END
DATE
FILMED
01-79
DDC

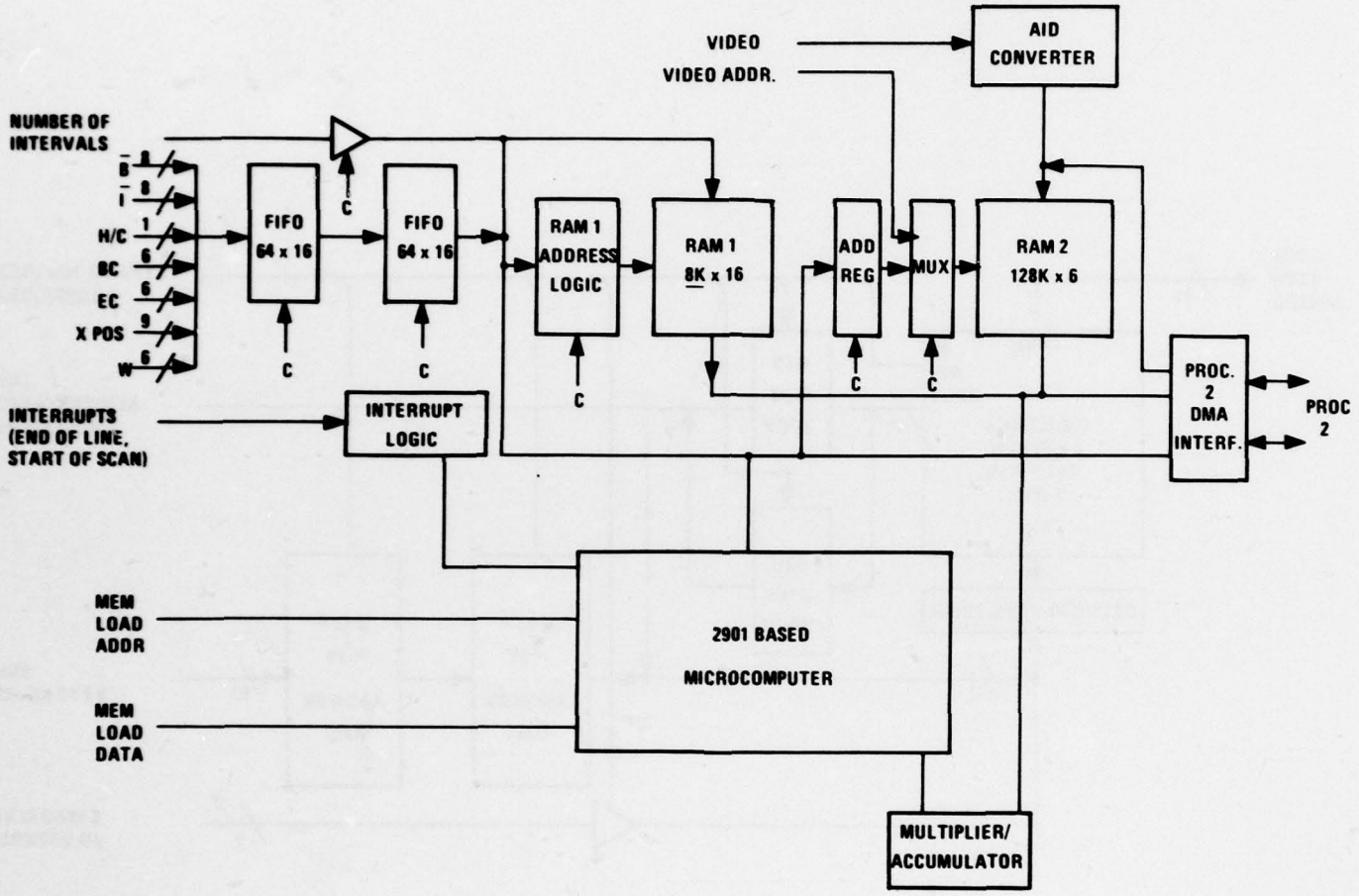


Figure 31. Computer System 1

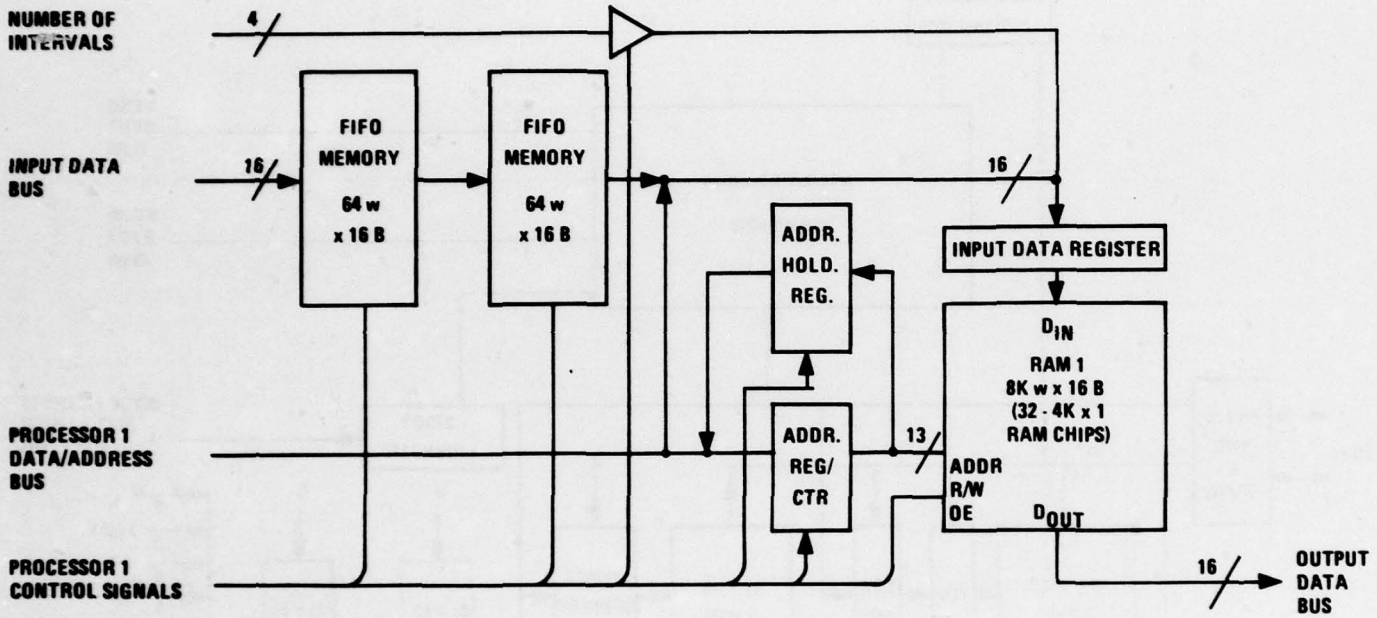


Figure 32. Memory 1 Configuration/Interface

TABLE 13. CPU 1

Inputs:

- 3- to 6-bit word at maximum rate of occurrence every 325 nsec (every 5 clock periods) from first level features
- Intensity data 6 to 8 bits in magnitude

Outputs:

- Must transfer position and target classification data to CPU 2

Processing:

1. Access Memory 2 with average cycle time of 200 nsec
2. Moment calculation
3. Use Memory 1 for scratch pad

Technology:

- Bipolar microprocessor--AMD2900 family
- Multiplier--TRW

Interfaces:

- Provide control for FIFO based upon an end of interval interrupt and an end of scan line interrupt
- Provide handshake to Memory 2
- Allow for DMA access to Memory 1 and Memory 2
- Provide interface to multiplier

Memory 1 Size:

8 K x 16 for first level feature
8 K x 16 for scratch pad and secondary feature
8 K x 16 for k-nearest neighbor prototype

24 K x 16 bits

CPU 2

The CPU 2 with symbol generation shown in Figure 33 is a commercially available 16-bit computer system. Its function is to do the interframe analysis, generation of symbol software, and training data gathering and diagnosis.

It is anticipated that this computer will be a DEC LSI 11/2 so that it will fit inside the box chosen for the rest of the hardware. The inputs to the operational software will be position and target classification. The output will be position information and data for loading symbol memory. The symbol memory will be loaded during vertical retrace. The parts to be included for the CPU 2 are:

1	KD11-HC	11/2 with 16 K Memory
2	KEV11	Fixed and Floating Point--Instruction Set
3	MRV11-AA	4 K PROM/ROM Board--Instruction Set
4	DLV11	Serial I/O Port
5	REV11-C	Refresh Bootstrap Diagnostic Board
6	DCK11-AC	LSI Interface Kit (2)
7	H9281-BB	Backplane/Card Guide--8 slots
8	RXV11-BA	Dual Floppy Controller
9	LSI20-HE	Decwriter (Keyboard/Printer) with cable
10	QJ013-CY	RT-11 Software
11	QJ813-CY	Fortran IV Software
12	QJV11-CB	PROM Formatter Software

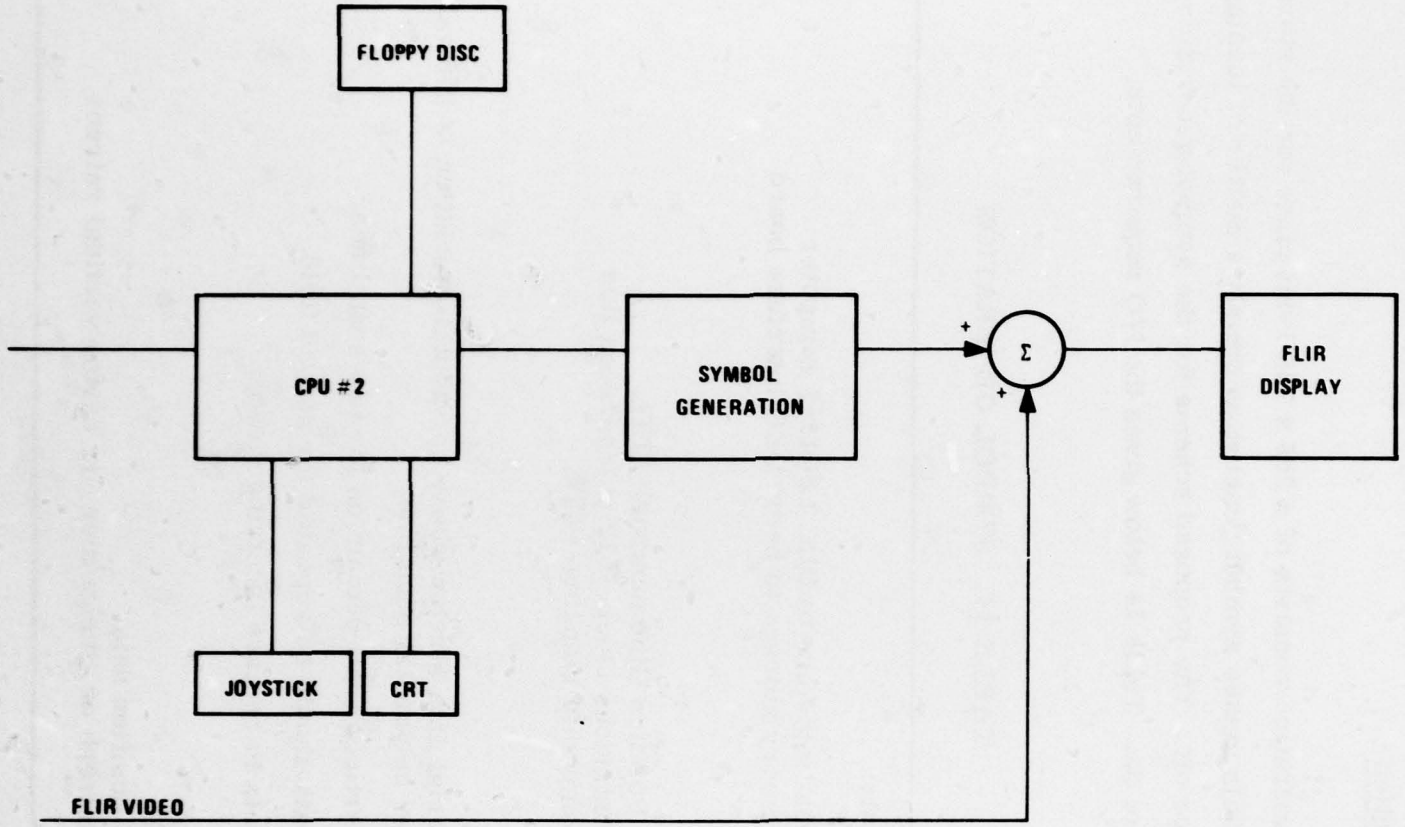


Figure 33. CPU 2 Configuration

Symbol Generation

The symbol generation consists of a 256 x 256 1-bit plane for the storage of the symbol. This makes symbol generation simply a matter of turning a set of bits on or off. The proposed scheme for the symbol generation is shown in Figure 34. Table 14 below gives the I/O requirements.

TABLE 14. SYMBOL GENERATION

Requirements:

1. Must interface to DEC LSI 11/2 computer
2. Memory storage to be on DEC interface board

Inputs:

- Y Position (line number)--TTL
- Continuous clock--512 pulses/scan line
- Composite blanking--TTL

Output:

1. Serial data stream whose horizontal resolution is 256 values per horizontal scan line
2. Vertical shall remain on for two scan lines.
3. Data should be repeated for second field.
4. Data to replace incoming video

Derived:

- X position data
- Refresh or change memory during vertical retrace

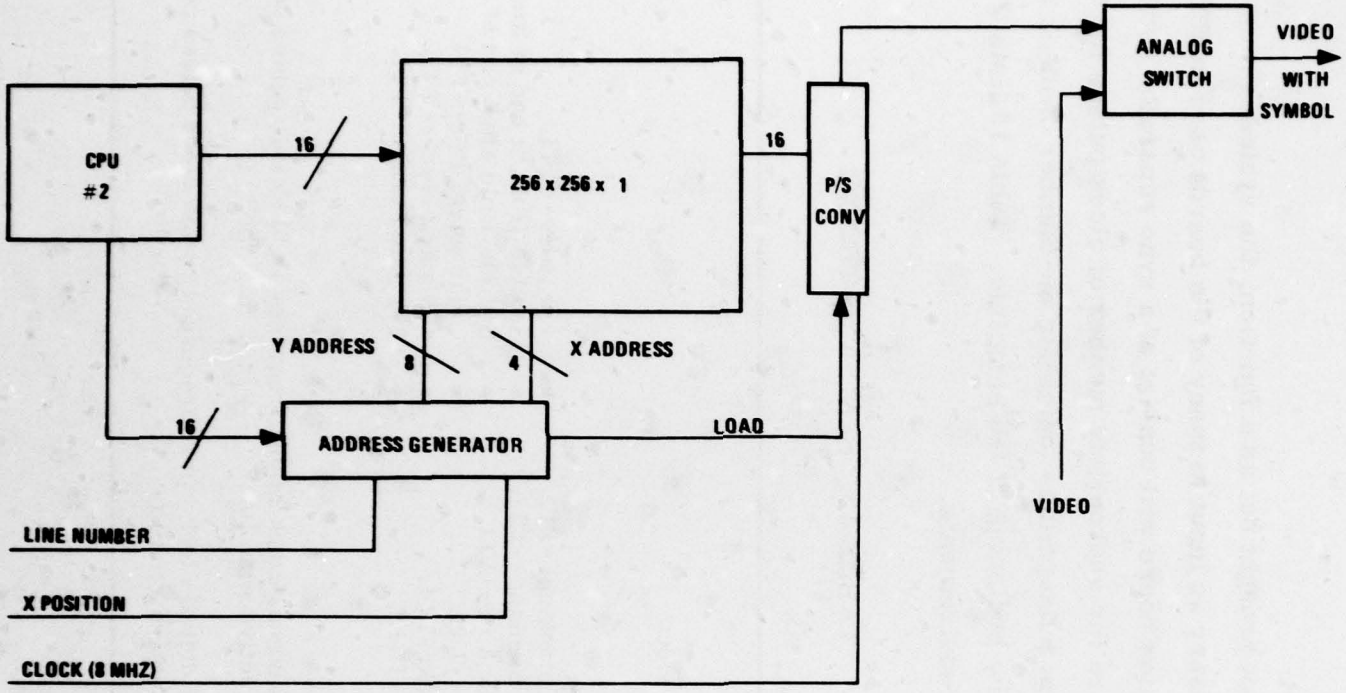


Figure 34. Symbol Generation

SYSTEM TIMING

Even though it is not brought out as a function, the system timing shown in Figure 35 is necessary as input to many of the boards used in PATS. Primarily, the timing board will consist of a sync separator and gated oscillator which give the appropriate number of clock pulses for the CCDs and give appropriate pulse widths indicating the number of clock pulses counted out since the beginning of the scan line. Table 15 lists the sync or master timing requirements.

TABLE 15. SYNC AND TIMING

Inputs:

- Composite video

Outputs:

- Composite blanking--TTL
- Horizontal blanking--TTL
- Vertical blanking--TTL
- CCD clock--TTL
- Digital clock--TTL
- 512 pulse--TTL
- 455 pulse--TTL
- Sample/Hold at end of line--TTL
- Sample/Hold after end of line--TTL
- Clear integrator--TTL
- 455 gated clock--TTL
- 512 gated clock--TTL

Derived within:

- Continuous digital clock whose frequency is 512 clock pulses per horizontal active scan line
- Continuous digital clock whose frequency is 455 clock pulses per horizontal active scan line

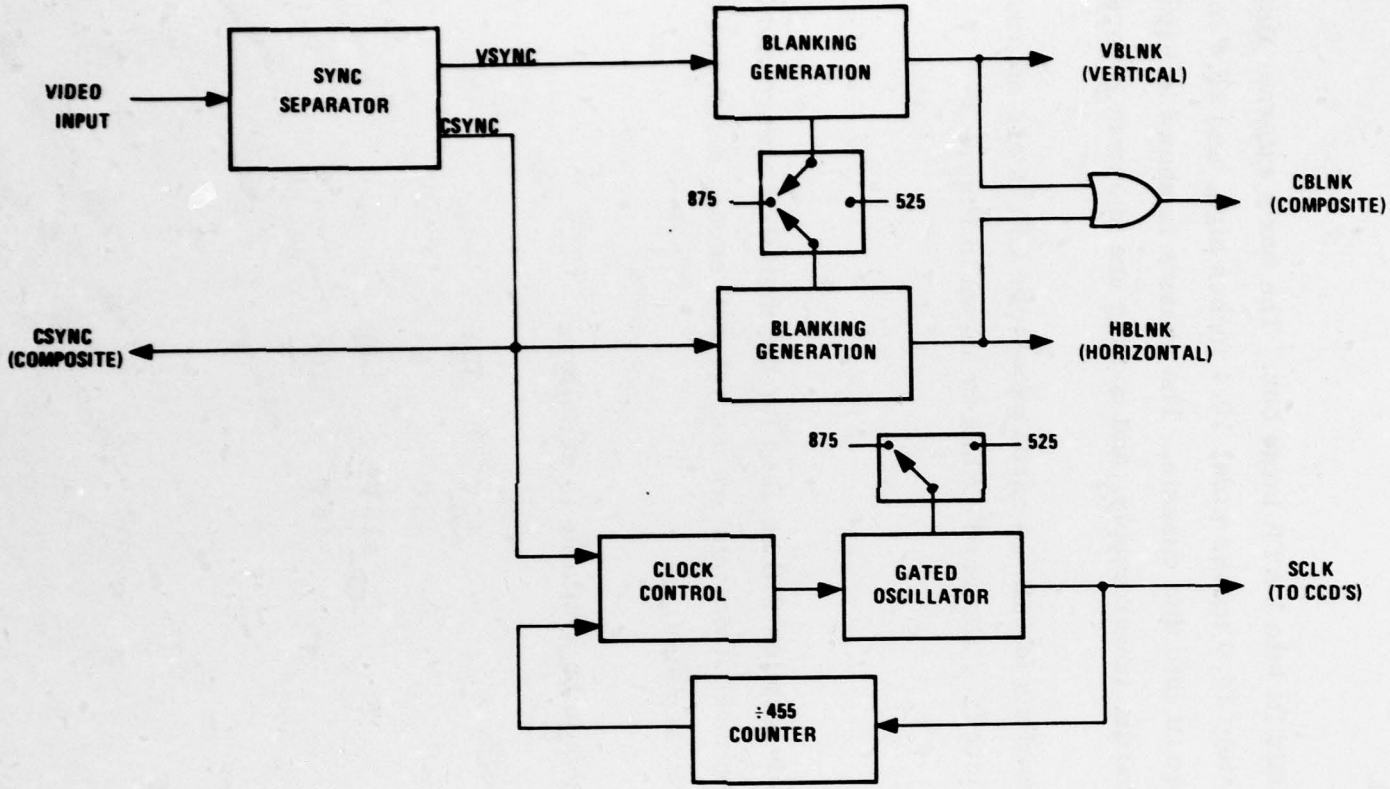


Figure 35. System Synchronizing and Timing

MECHANICAL

The PATS should fit into 1 ATR large box. The box's external dimensions are approximately 10.0 inches wide, 10.4 inches high, and 19.4 inches deep. CPU 2 will also fit into this chassis. The chassis is shown in Figure 36; the layout of cards, power supply, and cooling are shown in Figure 37.

The card dimensions for all the cards except the CPU 2 are approximately 9 inches x $6\frac{1}{4}$ inches. The card outline is shown in Figure 38.

POWER

Switching power supplies will be used for the digital hardware and CPU 2. The analog power requirements are not yet well enough defined to determine the linear power supplies.

Power which should be available is as follows:

+5V	35A
-5V	4A
+15V	10A
-15V	10A

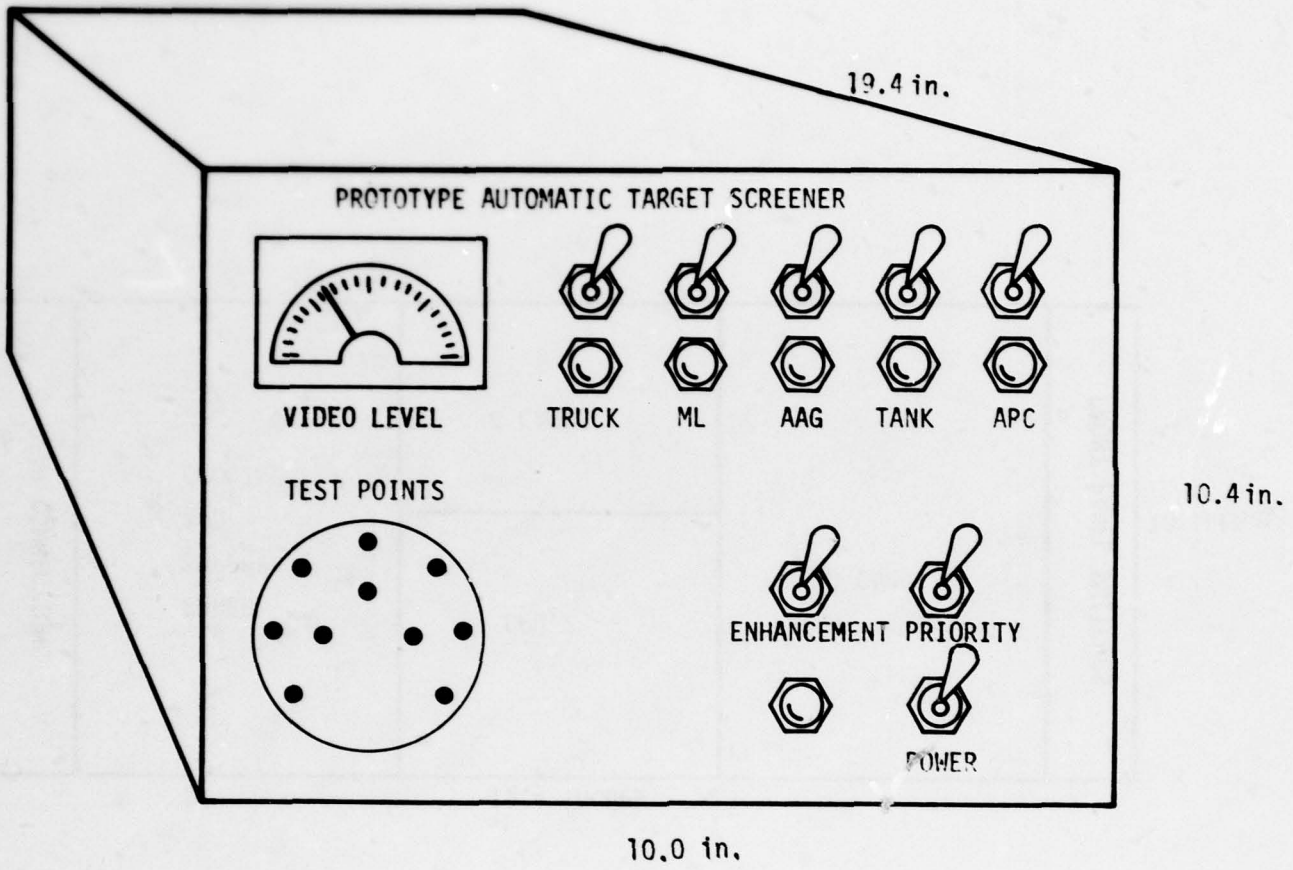


Figure 36. PATS Physical Configuration

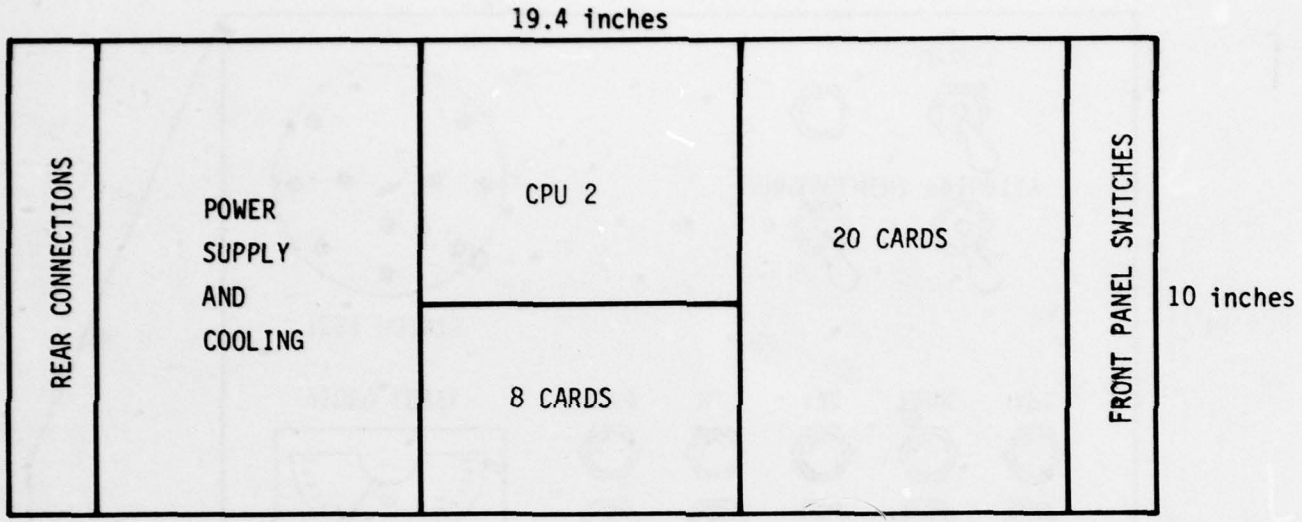


Figure 37. PATS Physical Layout

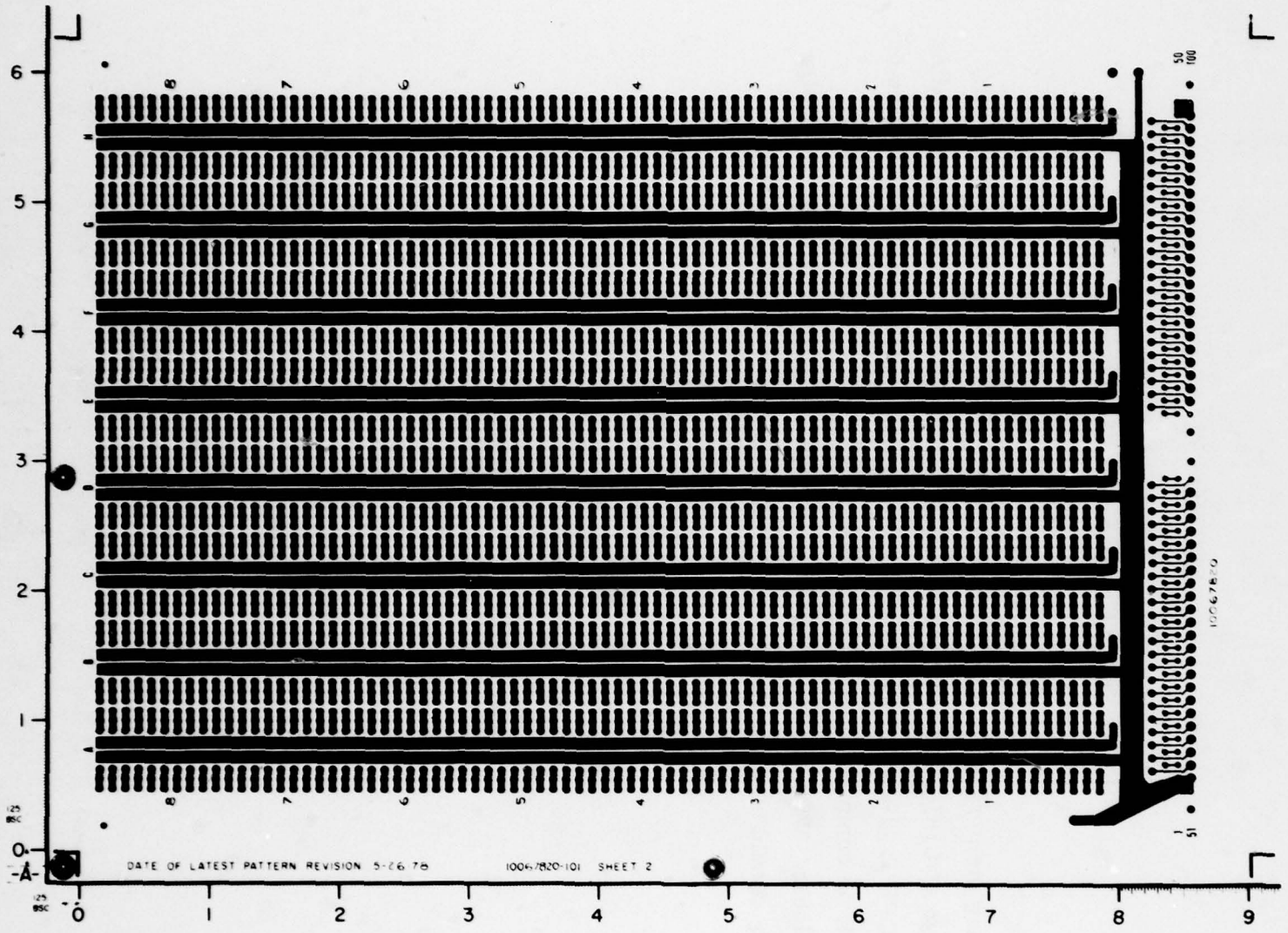


Figure 38. Card Outline

SECTION VI

PLANS FOR THE NEXT REPORTING PERIOD

During the next three-month reporting period, we plan to complete the design of CPU 1 and Memory 1. We will also begin microprogram coding for CPU 1. Detailed design of the feature extraction circuitry for Memory 2 will be completed and all boards will be fabricated as parts become available. With the exception of the symbol generation board, all design tasks should be essentially completed during the reporting period.